UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral thesis by

Shrirang K. Karandikar

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Prof. Sachin S. Sapatnekar

Name of Faculty Adviser(s)

Signature of Faculty Adviser(s)

2004-12-14

Date

GRADUATE SCHOOL

Synthesis and Performance Prediction of VLSI Designs

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Shrirang K. Karandikar

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Sachin S. Sapatnekar, Adviser

December 2004

Acknowledgments

This thesis is the direct result of the support and encouragement of my friends, family, and teachers. I would like to thank them all for their advice and help over the years.

The most important person from the above group is of course, Sachin Sapatnekar, who guided me in matters both academic and personal. He has struggled to teach me how to be meticulous in research and how to have clarity in thought and presentation (the jury is still out on whether these lessons have been learnt or not!). His insistence on precise writing has, I hope, rubbed off[1]. I equally value his support through what were difficult times for me; other people preach about being kind to others, helping people in trouble, etc., Sachin actually does. This is a lesson that I have definitely learnt.

Thanks also to members of my defense committee, Profs. Sobelman, Bazargan, Janardan and Reiner. I have learnt a lot more than mathematics from Vic, in whose classes I realized that demonstrating abstract concepts using simple examples (his approach) is more effective than pontificating about them (my usual approach). Prof. Janardan's course in algorithms cleared up a lot of ideas that I thought I knew, and introduced me to a lot of others. It also brought home the effectiveness of going over a subject systematically versus attacking it in a haphazard manner. This is a lesson that I have applied with great success to my research as well. I did not have the good fortune of taking courses under Profs. Sobelman and Bazargan, but did have the joy of interacting with them in seminars and informal discussions. Prof. Kia's suggestions during group meetings have been especially helpful.

I've had great times at my internships with IBM Research, and the experience of working with my mentors, Prabhakar Kudva and Chuck Alpert, have been invaluable. The enthusiasm that Prabhakar, Chuck and others I interacted with at IBM was infectious, and I look forward to further collaborations with them in the future.

A large contribution to the fun-filled times at Minnesota is due to the friends I made here. I treasure the many discussions (a few of which were also technical!) with

[1]The initial draft of this section referred to the set U, of all the people that I wanted to thank, which was composed of intersecting sets of T (teachers), F_1 (friends), F_2(family), etc.

Zhao Min, Jiang, Suresh and Hi!...hua. In addition to long chats about any and all subjects, Cristi and Tianpei were game for almost any crazy hiking/canoing/horse-riding trips. Friends who enriched my life in Minnesota and continue to do so include Saurabh, Nandan, Kaustubh, Kevin and AJ, and “The Bania” (the first and last were the best roommates anyone could have ever hoped for). Thank you all for the good times, and support through the tough times.

Friends from Clarkson University and Intel had a big role in helping me decide if I actually did want to pursue a doctorate, and where I should eventually go. These include Rajul, Gundu, their respective (and ever-expanding) families, Ashok and (ahem!) Khasim. At Intel, I had the good luck of meeting Tanay Karnik and Prashant Saxena, both of whose advice was invaluable. Peichen Pan, who guided me through my initial forays into EDA gently pushed me into applying to the right places. I thank Chris and Valerie Linstrum for their friendship over the years, and for making me feel at home, when I was away from home. Sudeep has been a friend since the last 17 years; here's looking forward to the next 17^n, with $n \to \infty$. Finally, Pradhan-kaka's ‘timely interventions’ were invaluable, in more ways that I can count.

This thesis is due to the unwavering faith and patience my family has had in me. Varun does not realize it yet (I didn't at his age), but it is a rare thing to have parents who give us the freedom to explore our interests, and encourage us to pursue our dreams. My life has had its ups and downs from the time I started my PhD until now, and the biggest up of all has been meeting and marrying Amita. Everything is now complete.

I would not be where I am today without the blessings of my Gurus. I have been fortunate in meeting them when I did. They took me in, turned my life around, and put me on the right path. I hope and pray to remain forever true to their teachings.

For my parents, who encouraged me to dream,
For Varun, the dreamer,
And for Amita, my dream come true.

Abstract

Expensive design closure issues for large VLSI circuits are caused by design flows that consist of steps that are independent of each other. The effect of choices made at different steps on the overall design are ignored. This thesis presents new algorithms that bridge the gap between early and later stages of the design, by focusing on two areas, early estimation of performance, and improved mapping algorithms.

A number of time consuming optimizations are carried out at different stages of the design process to improve performance. However, the improvements that can be achieved are indeterminate *a priori*. We address the issue of predicting performance gains in the context of gate sizing, by developing an algorithm for fast estimation of possible delay improvement without actually applying the sizing optimization. This approach is extended to determine the area overhead required for achieving a specified delay, which allows for comparisons of implementations based on the trade-offs between area and delay.

We next present a new algorithm for library-based technology mapping that incorporates sizing. The circuit is considered in its entirety, rather than in terms of fanout free trees, which leads to superior solutions when compared to the traditional approach. Finally, we address the problem of mapping to Silicon-On-Insulator (SOI) technology, used for implementing high performance designs. Algorithms developed for bulk CMOS can lead to suboptimal SOI circuits. We present a new algorithm for library-less technology mapping that take into account the peculiarities of SOI, and present the improvements obtained by our approach.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

The International Technology Roadmap for Semiconductors (ITRS) is cited as justification for a wide variety of theses, and this one is no exception. In the latest edition [ITR03], the Design Technology Working Group makes the case for the cost of design as being the greatest threat to the continuation of the semiconductor roadmap. Due to advances in semiconductor technology, the number of available transistors scales by a factor of two at each technology node, leading to a corresponding increase in design complexity. This requires at least a two-fold increase in productivity per node, without which the cost of implementing a design – in terms of engineer-hours, CPU-cycles, etc., – can quickly become unmanageable. One of the trends identified in [ITR03] as an approach to rein in the cost of design, is to use higher levels of abstraction in the design process. While this enables the design and implementation of circuits of increasing complexity, an immediate consequence is the increased separation of design representations during implementation. This effect is already evident: RTL typically has little information of physical characteristics such as interconnect parasitics, which can eventually have a drastic effect on performance parameters such as area, delay and power. Thus, the greater the separation between representations of a design, the greater the need for metrics that can predict the performance of the design at different stages of the design process, so as to avoid hidden surprises at the end of the process. This thesis presents metrics that predict possible performance improvements at specific stages of the design flow, hence allowing a designer better control over the design process.

In the current design methodology, individual stages of the design process are discrete; improvements in separate stages (such as a better synthesis approach, or a more efficient routing tool) are difficult to evaluate in a global setting. In particular, while a certain transform may improve the design *at that abstraction*, it may adversely impact the final implementation. A designer can conceivably choose to store a number of implementations, and defer the decision as to which one is the best to later stages. However, the multitude of options that are available at different stages of the design process lead to a correspondingly large number of implementations. Additionally, as we will show later, evaluating even a small number of implementations can be prohibitively expensive. Thus, the fundamental issue that has to be addressed, in order to support the implementation of designs of increasing complexity, is the need for metrics that can accurately predict the effects of various transformations in the global setting. Such metrics can enable a designer to make decisions that may be locally sub-optimal, but which are globally optimal. They can also be used to guide the design process, so that individual transformations are geared towards improving the performance of the final design, rather than the performance at the current level of abstraction.

In this thesis, we use the *delay* of a design as the performance measure, and study how it is affected by different transforms. The transforms that we focus on are *gate sizing* and *technology mapping* (which will be described shortly). We present techniques for estimating possible delay improvement under the sizing transform. We consider the cost (in terms of area) overhead for this delay improvement, and show how this cost can be estimated quickly, and with high-fidelity. Using this estimation approach, a designer can quickly determine the best implementation available. We present a new approach to technology mapping, and use our estimation technique to guide the mapping algorithm towards the best delay-optimal solution. Another aspect of technology mapping is that traditional approaches are no longer adequate when the implementation technology is drastically different. We address this issue under the context of silicon-on-insulator (SOI), and propose an algorithm that solves an SOI-specific problem.

1.1 Current Design Methodology

Figure 1.1 presents a broad overview of a typical flow used for implementing digital designs. The process of going from specification to silicon implementation can broadly

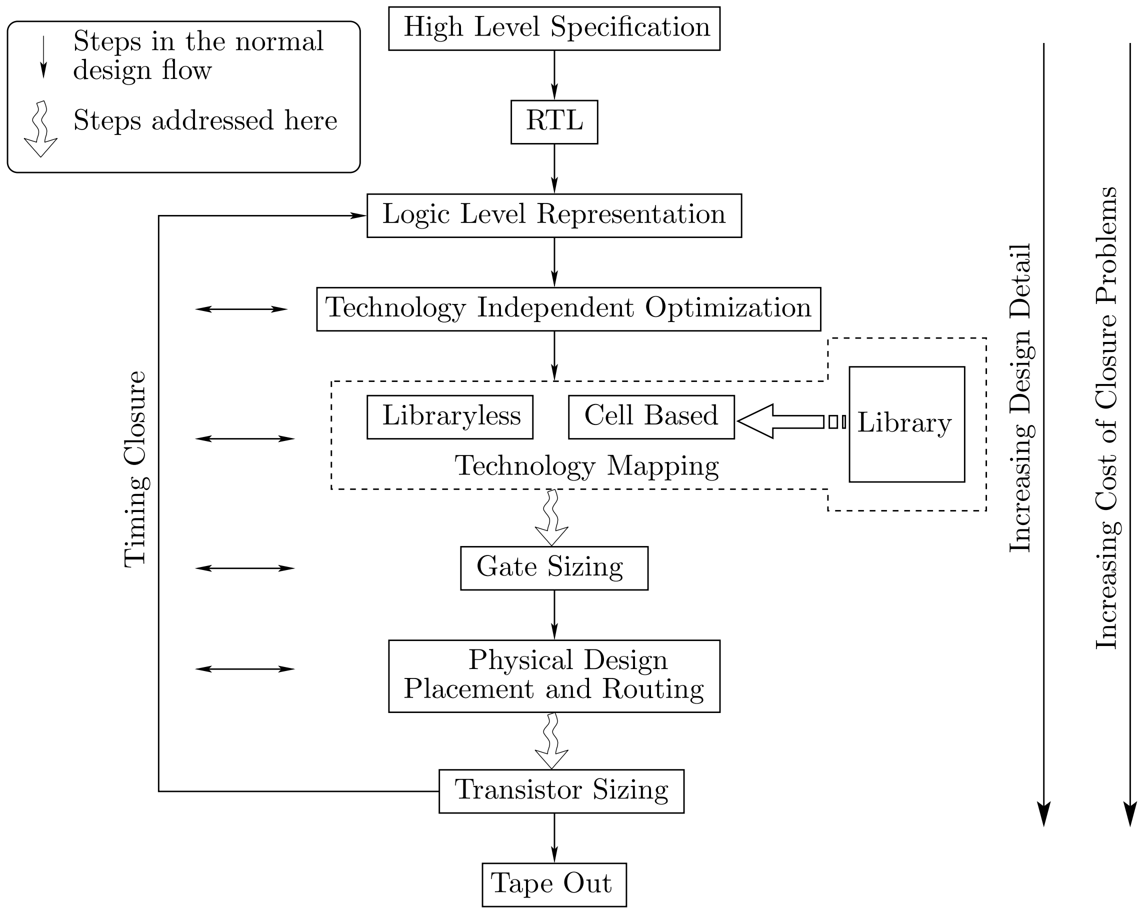


Figure 1.1: Typical Design Methodology

be viewed as one of successive refinement, from the initial specification and RTL implementation to a logic level representation, via *high level logic synthesis* (technology independent optimizations, technology mapping), followed by *physical synthesis and design* (placement, routing and gate sizing). At each of these stages, various optimizations are carried out, the goal being to improve some performance parameter, such as delay, area or power. The representation of a design changes as the implementation progresses through these steps. At the logic level, the design is in the form of a network of basic logic gates and their interconnections. This is a good representation of the *functionality* of the design, but cannot be used to determine the *performance* accurately. Technology mapping binds the logic level netlist to cells from a target technology library. The representation of the design is still in the form of a network, but the gates of the network are not arbitrary logic functions anymore, and specific characteristics of these gates (such as delay, area, power) are now known. At this level of abstraction, estimating the performance of the design is more accurate than before, but is still not close to the final value. This is because later stages can drastically alter the performance characteristics. For example, a placement and routing solution can change the load that a gate has to drive, which changes its delay (additionally, in today's small geometries, the interconnect delay itself is a significant component of overall delay).

Thus, as we progress through the design flow, more information about the design is available, which allows for more accurate estimation of the performance. However, this means that if a design is *not* going to meet its performance targets, this will be known only towards the end of the design process. In this case, the designer has to move back to previous stages of the design, and try to rectify this problem, by either re-implementing parts of the design or re-optimizing it, and the subsequent steps have to be repeated, with no guarantee of meeting the performance goals. This makes the design flow cyclical, rather than linear. These iterations are expensive in terms of time, and can lead to significant delays in completing a product.

The problem is that we would like to estimate the performance of a design in the early stages of the design flow. However, information that would allow the designer to make this estimation is not available until later stages, at which time it may be too late. This usually leads to over-designing at early stages, leading to cost-inefficient designs. In this thesis, we present new approaches that allow for early estimation, thus helping design closure and avoiding unnecessary cost overheads.

1.2 Previous Approaches

Various approaches that address the problem of early estimation have been suggested. In [PAB$^+$00] and [CKL$^+$03], a new methodology is presented, which tries to take cognizance of physical design effects in all stages of the design. This leads to more accurate optimizations, e.g., since synthesis is made 'placement-aware', any transform that changes a circuit updates the placement solution immediately. Accurate evaluation of any changes in the design is possible, since wire loads (and hence gate and interconnect delays) can be estimated from the placement. The obvious drawback of this approach is the run-time; reasonably large designs need more than a few days for synthesis to complete. More importantly, this approach is not a truly integrated one, in that the neither the high-level optimizations nor the placement algorithm are driven by the effect on each other, instead, individual effects are simply evaluated early and more often than in the traditional design flow. Thus, this methodology reduces the large iterations of Figure 1.1 to multiple local iterations.

In [SMCK01], a new approach called "Correct-by-Construction" is proposed, in which high level assumptions are used as specifications for later steps in the design flow. Thus, the focus switches from optimality to predictability, breaking the loops seen in Figure 1.1. However, the trade-off involved is not clear – the total time for implementing a design improves, but the potential performance gains that have been sacrificed are not quantified. In some sense, this approach is akin to selecting the "best" available implementation, and using this as the final solution, and hence is more useful in time-constrained designs, where time to market is more important than performance. Meeting performance targets for high-performance designs would be difficult with this approach.

Structured ASICs [MB02,PSS+03,HJLMS03] address the design productivity gap by targeting *regularity* in designs. Using regularity ameliorates problems associated with deep sub-micron designs, such as cross-talk and manufacture issues. As can be expected, there is a trade-off involved – achieving performance close to that obtained by a custom ASIC requires significantly higher area.

Other approaches that target early prediction for custom ASICs address specific optimization parameters, such as power and delay and build a design methodology that present the trade-offs between these parameters (e.g., [VS02]). The effect of specific physical manifestations that can impact design optimality has also been addressed, e.g., the interaction between placement on wire congestion [KSD02] and blockages on buffer insertion [HQGA02]. Specific high level estimation of delay [NN98], area [BN02], and power [NN96] have been proposed, but have the drawback of not considering physical design effects. Predicting improvements due to gate sizing, and load-aware technology mapping which is presented in this thesis, is an area that has been neglected until now.

1.3 Contributions of this thesis

We highlight our contributions in the context of the design flow presented in Figure 1.1. As mentioned before, we first consider the circuit sizing step, which is used to tune gate sizes in order to improve the delay along the critical path. Sizing is necessarily performed after routing, since the loads due to routing are known only after the fact. In Chapter 3, we present an fast and accurate algorithm that can predict what gains can be accrued using the sizing optimization, given a mapped, placed and routed solution. This allows the designer to evaluate a number of designs without actually having to perform gate sizing, which is usually an expensive, time consuming step. This approach provides an estimate of the minimum delay that a circuit can achieve under the sizing transform. We extend this approach in Chapter 4 to estimate the area cost of sizing a circuit to meet a given delay constraint. This allows a designer to make comparisons at intermediate delay points. Given a target delay, the smallest implementation that meets that delay will also have lower power consumption, and will therefore be the preferred solution. Our approach enables a designer to make such a selection.

Moving up the design flow, we address the problem of technology mapping. Current algorithms recognize the fact that the delay of a gate depends on the load being driven, and partly address this issue for single-fanout gates. However, these solutions are suboptimal when the entire circuit (which has gates with multiple outputs) is taken into consideration. Chapter 5 presents a new algorithm that performs technology mapping and correct sizing of gates for optimal delay. Finally, in Chapter 6, we consider the problems that occur due to the recent shift from bulk CMOS to silicon-on-insulator implementations. We show how using traditional approaches can lead to inferior (and in some conditions, erroneous) solutions, and present an algorithm that maps circuits while making use of the advantages allowed by SOI and avoiding the drawbacks.

In the current design flow, problems arise due to the iterations needed for timing closure. This thesis presents techniques that shrinks these cycles, by first allowing a designer to determine the ‘goodness’ of his design without having to perform expensive sizing optimizations, and second integrating technology mapping and sizing, leading to better quality implementations. These approaches can be extended to higher levels of abstractions as well, so that the designer has better insight into the performance of the design even earlier than technology mapping. The approaches presented here can also be applied to other design methodologies.

2 Logical Effort

As mentioned before, in this thesis we focus on the delay of the design as a measure of circuit performance. We use a simple model of delay called *logical effort*, described briefly in this section. A significant drawback of this model is in its handling of multiple fanouts in a circuit. We formalize this drawback as the "Load Distribution Problem", and present a solution in Chapter 3. Our improvement over the logical effort model is used for delay estimation in Chapters 3 and 4. A unique application to technology mapping is presented in Chapter 5.

The delay of a path of logic in a digital circuit is the sum of the delays of individual components (logic gates and interconnects) on that path. The delay of a circuit is the delay of the critical path, i.e., the maximum delay of any path from the inputs of the circuit to any output. As mentioned previously, our focus in this work is on estimating and optimizing the delay of a design, for which we need a good method of modeling the delay of individual gates.

The most accurate model of gate delay can be obtained by considering the transistor-level description of the gate at the layout-level, extracting all capacitive and resistive values, and using a SPICE-like formulation to determine switching characteristics. However, while this is the most accurate known method, it is extremely time consuming. It is straightforward to note that the delay of a gate is a function of the load that is being driven. Intuitively, the larger the load that has to be driven, the slower the gate is. We can therefore formulate the gate delay as a polynomial function of load. Higher-order models are naturally more accurate than simple linear ones. Another aspect of gate delay is the size of the gate itself. for a given load, the larger the gate is, the faster it will switch. In this work, we use a simple linear model of gate delay that captures the dependence of delay on output load (or load capacitance), gate size (or equivalently, input capacitance) and gate functionality. This approach is called the method of logical effort, and has been widely used in a variety of application domains [BKKS98, SIS99, DKS+00, SNvGK02] as well as in industry standard EDA synthesis tools [SKB+96, Mag02].

Using logical effort, the delay of a gate with input capacitance c_i is estimated by modeling it as a linear function of the load c_l being driven as:

$$\begin{aligned} d &= g \times \frac{c_l}{c_i} + p \qquad (2.1) \\ &= g \times h + p \\ &= f + p \end{aligned}$$

where

Logical Effort (g) is the complexity of the gate, relative to an inverter. It measures how much worse the gate is at driving a specified load than an inverter. The base case of an inverter is taken to have unit logical effort, and complex gates such as NAND, NOR and XOR have successively higher values of logical effort.

Electrical Effort, or Gain ($h = \frac{c_l}{c_i}$) describes how the electrical environment of the logic gate affects performance and how the size of the transistors in the gate determines its load-driving capability. c_l is the load being driven and c_i is the input capacitance of the gate under consideration.

Gate Effort ($f = gh$) is the product of the logical and the electrical efforts of the gate.

Parasitic Delay (p) expresses the intrinsic delay of the gate due to its own internal capacitance, and is largely independent of the size of the transistors in the logic gate.

The importance of this model (as compared to other delay models where the delay is a function of the load) is that it separates the different components that contribute to the delay of a gate in a manner that can be used for estimating minimum delays and sizing paths of logic, as will be described shortly. Another point to note is that the actual delay of the gate, is obtained by taking the product of d and τ, which is an implementation dependent parameter, that varies according to the technology.

2.1 Estimating the Minimum Delay

Equation (2.1) can be used to determine the delay of a gate if its electrical environment (output and input capacitances) is known. If we are interested in the delay of a path of logic, it can be obtained by summing the delays of each gate on that path (an operation that is similar to the approach using other delay models) as follows

$$\begin{aligned} D &= \sum d_i \\ &= \sum g_i h_i + \sum p_i \end{aligned} \tag{2.2}$$

where d_i is the delay, g_i the logical effort, h_i the electrical effort and p_i the parasitic delay of the i^{th} gate on the path.

However, the delay of a given path of logic can change if the sizes of the gates are varied. For example, increasing the size of an intermediate gate will make it faster, but this increases the load on its input gate, hence slowing it down. The net effect may be an improvement or a degradation in the path delay, depending on the gates involved and the change in size. This leads naturally to the question of what is the smallest value of path delay, if gate sizes are allowed to change. This is where logical effort is most useful, as follows.

Similar to the gate logical and electrical efforts, we can also define the *path* logical effort

$$G = \prod g_i \tag{2.3}$$

and *path* electrical effort

$$H = \frac{C_{out}}{c_{in}} \tag{2.4}$$

where C_{out} is the load being driven by the output gate of the path, and c_{in} is the input capacitance of the first gate on the path.
the *path effort*, F, for path of logic is simply the product of the path logical and electrical efforts (this is similar to the relation between gate effort, gate logical effort and gate electrical effort)

$$F = G \cdot H \tag{2.5}$$

the delay of a path of logic is minimized when the path effort is distributed equally over all gates in the path (this is proved in [SSH99]). this happy state of affairs occurs when

$$f_i = g_i h_i = F^{\frac{1}{N}} \tag{2.6}$$

for a path with N stages.

If each stage on a path has the same effort, Equation (2.2) can be re-written as

$$\hat{D} = NF^{\frac{1}{N}} + P \tag{2.7}$$

where $\hat{D}$ is used to represent the minimum delay.

Thus, given a path of logic, logical effort can be used to easily determine the minimum delay achievable by that path. This computation is efficient, since a single traversal is enough to determine all required parameters. We note that this model is fast at the expense of accuracy. These inaccuracies are due to the simple linear model, but are within permissible limits for the purposes of this work.

2.2 Sizing for Minimum Delay

Once the minimum delay is determined, the individual gate sizes that achieve this delay can be recovered in a straightforward manner as follows. Recall that each gate now has an effort $\hat{f} = F^{\frac{1}{N}}$. Since $\hat{f}$ is also equal to gh, we have

$$\begin{aligned}
\hat{f} &= F^{\frac{1}{N}} \\
&= g \times h \\
&= g \times \frac{c_{out}}{c_{in}} \\
c_{in} &= \frac{g \times c_{out}}{\hat{f}} \qquad (2.8)
\end{aligned}$$

The output gate of a path drives a load C_{out}, hence starting at the output gate, we can successively determine the gate sizes by traversing the path to the input. At the end of such a traversal, we will have obtained gate sizes such that the path has a minimum delay given by Equation (2.7). In current design geometries, interconnect is an increasing component of load. In [SSH99], extensions have been proposed to accommodate interconnect capacitance into the parasitic component of delay. For long segments of interconnect, the traditional approach is to add buffers at intermediate points, logical effort can then be used as before to analyze this circuit.

2.3 Problems in Paradise

For simple paths of logic, where every gate only drives one gate at its output, Equation (2.7) can be used to determine the minimum delay that is achievable, and Equation (2.8) can be used to obtain the corresponding gate sizes. However, the biggest drawback to the method of logical effort occurs when gates drive multiple fanouts, which is often the case in realistic circuits.

The heuristic used to account for multiple fanouts is to add a new parameter called the *branching effort*, b for a gate, defined as

$$b = \frac{c_{total}}{c_{useful}} \qquad (2.9)$$

where c_{total} is the total load being driven, and c_{useful} is the capacitance on the path of interest. The branching effort along a path , B, is then

$$B = \prod b_i \tag{2.10}$$

where b_i is the branching effort of gate i as calculated in Equation (2.9). Of course, if a gate has only one fanout, its branching effort is unity.

The gate effort (Equation (2.1)) and path effort (Equation (2.5)) are then modified in order to capture the effect of off-path fanouts as follows

$$f = gbh \tag{2.11}$$

$$F = G \cdot B \cdot H \tag{2.12}$$

The analysis of a path is then carried out as before, except with the modified values of efforts. This approach has a few serious drawbacks. Note that only one path of the circuit is analyzed at a time. In a well designed circuit, a large number of paths will be close to critical, and using this approach can quickly become impractical. Another important issue is that this approach compels non-critical fanouts to contribute a load that scales in proportion to the load of the fanout of interest, which is determined *only* by gates on the path being analyzed. This has two disruptive effects. First, the load on the path under consideration, and hence its delay estimate, is larger than necessary, due to the contribution of the non-critical fanouts. Second, assumptions of gate sizes on the path being analyzed affect the loads on the other gates in the circuit. This may lead to the delay of a different path becoming dominant. The branching factor does not capture this interaction among different paths in a circuit. Analyzing every path, and the interactions among all paths is not feasible, because of the exponential number of such paths in a circuit. Thus, while the method of logical effort is well suited to analyze single path delays, it cannot be used directly when critical paths are not well defined, or can change. In the following chapter, we present an approach that can handle such scenarios.

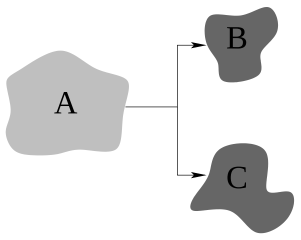


Figure 2.1: Assigning Capacitance at Multiple Fanouts

2.4 The Load-Distribution Problem

We now formalize the problem that the branching factor tries to solve. Consider the situation shown in Figure 2.1, with some logic A having two fanouts, B and C, which eventually drive primary outputs. Each of A, B and C are fanout-free regions of the circuit. The delay of segment A depends directly on the load being driven, which, in this case, is the input capacitance of B and C. There are two situations that have to be considered:

The interaction between A **and its outputs** Assigning a larger input capacitance to B and C makes them faster, at the cost of increasing the load on A, and slowing it down, and vice versa. What is the optimum value of capacitance that should be assigned to the output of A, so that the delay of the *entire* circuit is minimized?

The interaction between B **and** C These two fanout-free regions may have completely different delays to the primary outputs of the circuit, which are influenced by the constituent logic and their respective input capacitance. If this is the case, we would like the critical branch to have a larger input capacitance. On the other hand, if the output branches have the same delay, they should have the same input capacitance. Thus, if we determine the optimal load that A should be driving, what is the best distribution of this capacitance to each fanout?

We refer to these two problems together as the *load-distribution problem.* This formalization does not use the heuristic branching factor from logical effort, and captures the effect of multiple, interacting paths between primary inputs and outputs.

A solution to this problem is presented in the following chapter, and this solution is flexible enough that we can apply it, in Chapter 5, to technology mapping as well.

Given a load at a multiple fanout point in the circuit, current algorithms can determine the best mapping for the logic up to that point. However, this load is typically estimated using heuristics, and since the mapped solution depends directly on the load being driven, wrong estimates can lead to sub-optimal solutions.

3 Fast Estimation of Minimum Achievable Delay using Circuit Sizing

As mentioned in Chapter 1, designs can be mapped to different implementations using diverse approaches, with varying cost criteria. Post-processing transforms, such as transistor sizing can drastically improve circuit performance, by optimizing critical paths to meet timing specifications. However, most transistor sizing tools have high execution times, and the attainable circuit delay can be determined only after running the tool. In this chapter, we present an approach for fast estimation of possible benefits due to transistor sizing that can enable a designer to choose one among several functionally identical implementations. Our algorithm computes the minimum achievable delay of a circuit with a maximum average error of 5.5% in less than a second for even the largest benchmarks.

3.1 Introduction

Implementing a design involves synthesis (technology independent optimizations and technology mapping), placement and routing. In a final timing correction step, transistors of logic gates are appropriately sized to speed up critical paths, thus incurring an area overhead for gains in circuit speed. The importance of transistor sizing can be judged by the amount of research carried out both in academia [FD85,SRVK93,CCW98,SSP02] and in industry [CCH^{+}98,BVSH02]. However, these optimization tools have large running times, and can take up to a few hours to calculate the appropriate solution for an industry-sized circuit. In this scenario, it is difficult for a designer to determine if an implementation will be able to meet performance goals after transistor sizing, or which circuit out of multiple different implementations for the same functionality should be chosen for future detailed optimization.

The *delay* of a circuit is the maximum delay of all PI-to-PO paths of the circuit. Transistor sizing is applied to the circuit to reduce this delay, in order to meet design goals. The smallest value of delay that can be obtained in this manner is referred to as the *minimum achievable delay*. In this chapter, we present an approach that can quickly estimate this minimum delay when transistor sizing is applied to a mapped circuit. Gains due to this optimization vary according to the circuit being sized, because of a number of factors, such as which logic gates are used, how these gates are connected, and how much load they drive. The minimum achievable delay captures how amenable a circuit is to transistor sizing. Thus, while circuits are rarely sized to the minimum delay value (due to the associated high area overheads), it is a good measure of circuit quality. Using our tool as a *fast* estimator of the minimum achievable delay, a designer can make early comparisons among different solutions provided by placement and routing for the same functionality, without incurring the cost of an actual sizing step. Once the designer has chosen one of the candidate implementations based on this metric, a more exact optimizer can be used to obtain actual transistor sizes.

Our approach is based on logical effort [SS91, SSH99], which has been described in Chapter 2. As mentioned there, logical effort is well suited for estimating the minimum achievable delay of a *single* path in a circuit, with a heuristic branching factor used to account for multiple fanouts. However, the critical path of a circuit changes dynamically according to the choice of distribution of capacitance over multiple fanouts, which is a constituent of the load distribution problem. An important contribution and differentiator of the algorithm presented in this chapter is an exact solution to the load distribution problem. This solution allows us a means of accurately determining the minimum achievable delay of a circuit by implicitly considering *all* paths of the circuit at the same time.

3.2 An Algorithm for Minimum Delay Estimation

Every gate has multiple sizes available, each of which corresponds to an input capacitance, C_{in}. We define $\mathcal{C}_{in_{\mathrm{G}}}$ to be the set of all possible values of C_{in} of a gate G. A gate may drive multiple fanouts, and the load capacitance being driven, c_l, is the input capacitances of these fanouts, combined with routing capacitances, c_r. For a gate G driving n fanouts, $\mathrm{F}_1, \mathrm{F}_2, \cdots \mathrm{F}_n$, the possible values of the load capacitance is described by the sum of c_r and the elements of the set $\mathcal{C}_{L_{\mathrm{G}}}$, defined as

$$\mathcal{C}_{L_{\mathrm{G}}} = \left\{ \sum_{j=1}^{n} c_j : \forall c_j \in \mathcal{C}_{in_{\mathrm{F}_j}}, j = 1 \cdots n \right\} \tag{3.1}$$

With this terminology, we now present a dynamic programming based approach for calculating the minimum achievable delay of a circuit. The basic approach is to traverse the circuit from primary outputs to primary inputs. For each size of a gate, we are interested in minimizing the maximum delay from the input of the gate to any primary output PO. This is achieved by computing a delay-C_{in} curve, each point of which is represented by $D_{\mathrm{G}\to PO}[c_i], \forall c_i \in \mathcal{C}_{in_{\mathrm{G}}}$, defined as the maximum delay from G, with size c_i, to any PO. This value is determined by first calculating the delay of the gate (represented by $D_G[c_{in_i}][c_l]$) for each load value obtained by $c_l = c_j + c_r, c_j \in \mathcal{C}_{L_{\mathrm{G}}}$, and then adding it to the delay (corresponding to the load) from the fanout of the gate to the primary outputs.

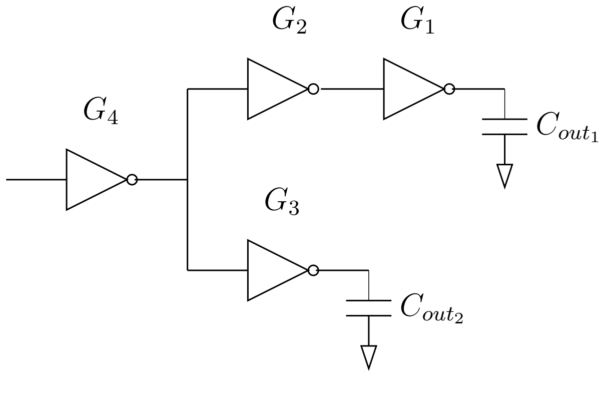


Figure 3.1: Delay-C_{in} Calculation and Propagation Across Multiple Fanouts

The details of our approach, which is provably optimal for trees, are presented in the following subsections. We first present the calculation for gates with single fanouts, and then show how this calculation can be extended to gates with multiple fanouts. We use the circuit in figure 3.1 to illustrate the discussion.

Single Fanouts

The output of a gate having a single fanout can be connected to a fixed load, such as at the primary output, or to another gate, in which case the load being driven depends on the size of the gate at the output. We present these two scenarios individually for ease of explanation; the fixed load is actually a special case of the variable load with only one load value.

1. **Fixed Load:** Consider gate[1] G_1 driving a load of C_{out_1} at a primary output as an example of this case, as shown in figure 3.1. Calculating the delay-C_{in} curve is straightforward; each possible gate size corresponds to a different value of $c_i \in \mathcal{C}_{in_{G_1}}$ for the gate, and the delay can be calculated using Equation (2.1). The delay to the primary output is the same as the gate delay in this case. Therefore,

$$\begin{aligned} D_{G_1 \to PO}[c_i] &= D_{G_1}[c_i][c_l] \\ &= [g \times \frac{c_l}{c_i} + \text{parasitic delay}]_{G_1} \\ \text{where } c_l &= C_{out_1} \end{aligned} \tag{3.2}$$

   For different sizes of G_1 in Figure 3.1, Plot I of Figure 3.2 presents the delay from the input of G_1 to the primary output. Since the load is fixed, we obtain monotonically decreasing values of delay for increasing gate sizes.

2. **Variable Load:** Next, consider gate G_2 driving gate G_1. The load seen by G_2 is not fixed, as in the previous case, but varies according to the size of G_1. The delay-C_{in} curve for gate G_1 has already been calculated. The delay-C_{in} curve

[1]In this discussion, all gates are shown as inverters for illustration purposes only. The method applies directly to more complex gates, with appropriate values for logical effort and parasitic delay.

for gate G_2 is calculated in a two-step procedure. First, for a particular gate size of G_2, corresponding to an input capacitance of $c_i \in \mathcal{C}_{in_{G_2}}$, we examine all sizes of G_1, and calculate the delay as shown in Equation (3.3). Here, the load c_l driven by gate G_2 is the input capacitance of gate G_1, and the parasitics of the wire connecting the output of G_2 to the input of G_1. Since there is a single gate connected to the output, $\mathcal{C}_{L_{G_2}} \equiv \mathcal{C}_{in_{G_1}}$.

$$\begin{aligned} D_{G_2}[c_i][c_l] = & \quad [g \times \tfrac{c_l}{c_i} + \text{parasitic delay}]_{G_2} \\ D_{G_2 \to PO}[c_i] = & \quad \min_{c_j \in \mathcal{C}_{L_{G_2}}} \{D_{G_2}[c_i][c_l] + D_{G_1 \to PO}[c_j]\} \\ \text{where } c_l = & \quad c_j + c_r \quad \forall c_j \in \mathcal{C}_{L_{G_2}} \end{aligned} \tag{3.3}$$

Next, the delay from the input of G_2 to the primary output is obtained by combining $D_{G_2}[c_i][c_l]$ with $D_{G_1 \to PO}[c_j]$, which is the delay corresponding to the size of G_1 under consideration. Thus, the minimum delay that we can obtain for the selected size of G_2 is determined using Equation (3.4). Note that this size of G_1 that minimizes $D_{G_2 \to PO}[c_i]$ may be suboptimal when G_1 is considered in isolation, i.e., it may not be the size that minimizes the delay from G_1 to the primary output.

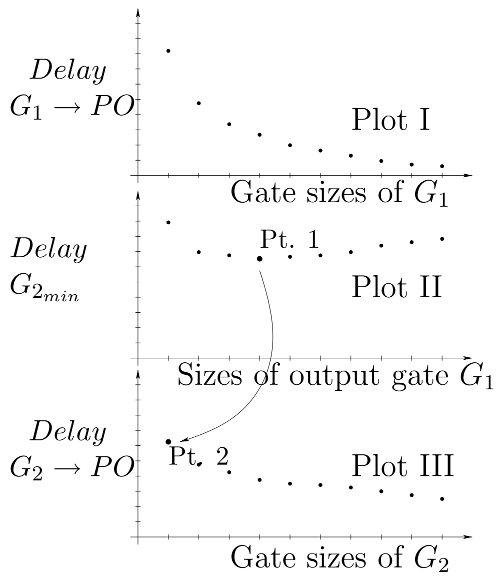


Figure 3.2: Delay-C_{in} Curve Propagation Across Gates G_1 and G_2 from figure 3.1

For a particular size of gate G_2, the delay from G_2 to the primary output varies due to two factors changing simultaneously, the load being driven by G_2 (which corresponds to different sizes of gate G_1) and the corresponding delays of G_1. Thus, for the selected size of G_2, there is a trade off between selecting larger sizes of G_1 (which reduce the delay of G_1 but slow G_2 down), and smaller sizes of G_1 (which have higher delays for G_1 but decrease the delay of G_2).

Equations (3.3) and (3.4) calculate the best solution for the selected size of gate G_2. This calculation is repeated for different sizes of G_2, to obtain its complete delay-C_{in} curve.

To illustrate this calculation, consider the smallest gate size of G_2. For this size, Plot II in figure 3.2 is the delay from G_2 to the output, for different sizes of G_1, corresponding to different values of load capacitance for gate G_2. The minimum delay value of this set is at the point labeled p1, and all other points that have higher delay can be discarded, as they are suboptimal.

For different sizes of G_2, the delay to the primary output is as shown in Plot III of figure 3.2. Point p2 is the delay for the smallest size of G_2, and is obtained from Plot II as described above. The remaining points in Plot III are obtained by repeating this calculation for all other sizes of G_2.

Multiple Fanouts

The scenarios presented above are the most basic cases, with a single fanout on a gate. We now consider the case when a gate drives multiple fanouts, such as gate G_4 in Figure 3.1. It is in our approach here that we differ the most with respect to the method of logical effort, since we take into account different values of delays and gate sizes on *each* fanout simultaneously. Depending on the sizes of the gates, either of the paths through gate G_2 and gate G_3 may have larger delays, and hence could be critical. Our formulation explicitly accounts for this changing dynamic.

In Figure 3.1, assume the delay-C_{in} curves of gates G_2 and G_3 have been calculated. In this case, $\mathcal{C}_{L_{G_4}} \equiv \mathcal{C}_{in_{G_2}} \times \mathcal{C}_{in_{G_3}}$. The load being driven by gate G_4, c_l, is the sum of the input capacitances of gates G_2 and G_3, and the routing capacitance c_r. For a particular size of G_4 (and a corresponding value of input capacitance $c_i \in \mathcal{C}_{in_{G_4}}$), we calculate the delay of gate G_4 for each value of this load as shown in Equation (3.4).

$$\begin{aligned} D_{G_4}[c_i][c_l] &= \{g \times \tfrac{c_l}{c_i} + \text{parasitic delay}\}_{G_4} && (3.4)\\ D_{G_4\to PO}[c_i] &= \min_{c_j,c_k}\{D_{G_4}[c_i] + D_{OP\to PO}\} && (3.5)\\ \text{where } D_{OP\to PO} &= \max\{D_{G_2\to PO}[c_j], D_{G_3\to PO}[c_k]\}\\ \text{and } c_l = c_j &+ c_k + c_r \quad \forall c_j \in \mathcal{C}_{in_{G_2}}, c_k \in \mathcal{C}_{in_{G_3}} \end{aligned}$$

Next, the delay to the primary output is calculated by combining $D_{G_4}[c_i]$ with the maximum of the delays of each fanout. In this scenario, the identity of the branch with the maximum delay to a primary output can change according to which branch has higher delay. For a particular size of G_4, one branch may determine the critical path, while the other may be critical for another size of G_4. The formulation of Equation (3.6) automatically accounts for this. Thus, for the selected size of G_4, we can determine the optimal size of each of its outputs, in order to obtain the minimum delay. This procedure is repeated for all sizes of G_4, to compute the entire delay-C_{in} curve for G_4.

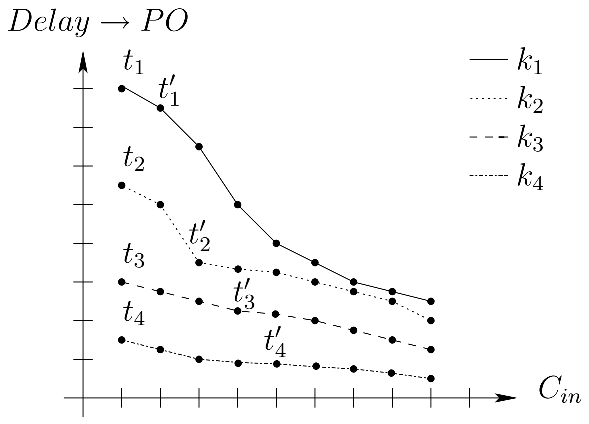


Figure 3.3: Combining Delay-C_{in} Curves at Multiple Fanouts

It may seem that the size of the set $\mathcal{C}_{L_G}$ for a gate G with multiple fanouts is proportional to the product of the number of sizes of the fanout gates. Assume gate G drives four outputs, whose delay-C_{in} curves are represented by k_1, k_2, k_3 and k_4, shown in Figure 3.3. If each fanout has m sizes, each curve has m points, and the size of $\mathcal{C}_{L_G}$ is m^4. However, we can show that most of the values in $\mathcal{C}_{L_G}$ are redundant. For example, consider the tuple $\mathcal{T}$ of the first points t_1, t_2, t_3 and t_4 from each of the curves in figure 3.3. A tuple $\mathcal{T}^{'}$ of the point t_1 from curve k_1 and any other point from k_2, k_3 and k_4 (say $t_2^{'}$, $t_3^{'}$ and $t_4^{'}$), is *inferior* to $\mathcal{T}$ for the following reason. There are two values that are extracted from $\mathcal{T}$ and $\mathcal{T}^{'}$, the maximum delay to a primary output, and the sum of the input capacitances represented by these combinations, which is used as the load in the delay calculation of gate G. The maximum delay is the same in tuples $\mathcal{T}$ and $\mathcal{T}^{'}$, but the load presented by $\mathcal{T}^{'}$ is greater than that of $\mathcal{T}$. Hence, the delay of G (calculated using Equation (3.4)), and therefore the delay to a primary output (calculated using Equation (3.6)) is larger in this case. Since we are interested in minimizing the delay to a primary output, the solution offered by tuple $\mathcal{T}^{'}$ will never replace that calculated using $\mathcal{T}$.

The above discussion directly leads to a strategy for efficiently selecting useful values of c_l from the delay-C_{in} curves of outputs. First, these curves are stored in order of non-increasing delay (and hence increasing sizes). The first c_l is the routing capacitance c_r plus the capacitance corresponding to the maximum-delay points from each curve, as in tuple $\mathcal{T}$. The next value is obtained by replacing the point with maximum delay (e.g., t_1 of curve k_1 in $\mathcal{T}$), with the next point from the same curve ($t_1^{'}$). This effectively ignores the combination of t_1 with remaining points from the other curves. This process is continued till the maximum delay point is the last point on its curve. Thus, the total number of combinations is of the order of the *sum* of number of points on each curve, rather than the product.

Algorithm

Algorithm 3.1, `Minimum_Delay_Estimation (MDE)`, presents our algorithm for estimating the minimum achievable delay of a circuit. The calculation is based on the delay-C_{in} curve computation presented in the previous subsection. All gates are processed in topological order, from POs to PIs. At each primary input, the data point corresponding to the minimum delay is selected, the maximum over all PIs is the desired minimum achievable delay.

Assume that there are m sizes for each gate in a circuit with N gates, and the

Algorithm 3.1 `MDE: Minimum_Delay_Estimation`

```
for each gate G whose outputs have been processed do
  // calculate the delay-$C_{in}$ curves for G
  for all $c_i \in \mathcal{C}_{in_\mathrm{G}}$ do
    $D_{\mathrm{G}\rightarrow PO}[c_i] = \infty$
    for every $c_j \in \mathcal{C}_{L_\mathrm{G}}$ that is not redundant do
      // G has n fanouts $\mathrm{F}_1, \mathrm{F}_2, \cdots \mathrm{F}_n$
      $c_l = \sum_{j=1}^{n} c_j + c_r$
      // determine the delay of gate G
      $D_\mathrm{G}[c_i][c_l] = [g \times \frac{c_l}{c_i} + \text{parasitic delay}]_\mathrm{G}$
      // determine maximum delay from any fanout F
      // to any PO, using the delay-$C_{in}$ curves of F
      $temp = D_\mathrm{G}[c_i][c_l] + \max_{j=1\cdots n}(D_{\mathrm{F}_j\rightarrow PO}[c_j])$
      $D_{\mathrm{G}\rightarrow PO}[c_i] = \min(temp, D_{\mathrm{G}\rightarrow PO}[c_i])$
    end for
  end for
end for
Minimum Delay = $\max\{\min_{\text{all PI's}}\{\text{delay to PO}\}\}$
```

maximum fanout on any gate is $|FO|$. The innermost `for` loop is executed $O(m \times |FO|)$ times, as shown previously, and the cost of determining the maximum delay point is $O(|FO|)$. The second `for` loop is executed m times, since we assume m sizes for each gate. Finally, since there are N gates in the circuit, the outermost `for` loop is executed N times. Thus, the running time of algorithm `MDE` is $O(N \cdot m \cdot m \cdot |FO| \cdot |FO|)$. However, note that this is a very loose upper bound, since very few gates actually have $|FO|$ fanouts.

Algorithm `MDE` is optimal for trees. However, most circuits are DAGs, with reconvergent fanouts. The main problem with DAGs is that there are multiple paths from a particular gate to primary outputs, or between two gates. An implicit assumption of our algorithm is that the delay-C_{in} curves at multiple fanout points are independent, and that we are free to choose the combination of output delays and capacitances that best suit the current gate. However, with reconvergent fanouts, these choices are not independent of each other. Selecting a data point on one output restricts the choices on the other, and determining the relation between different outputs is intractable for general circuits. However, assuming independence is not unreasonable.

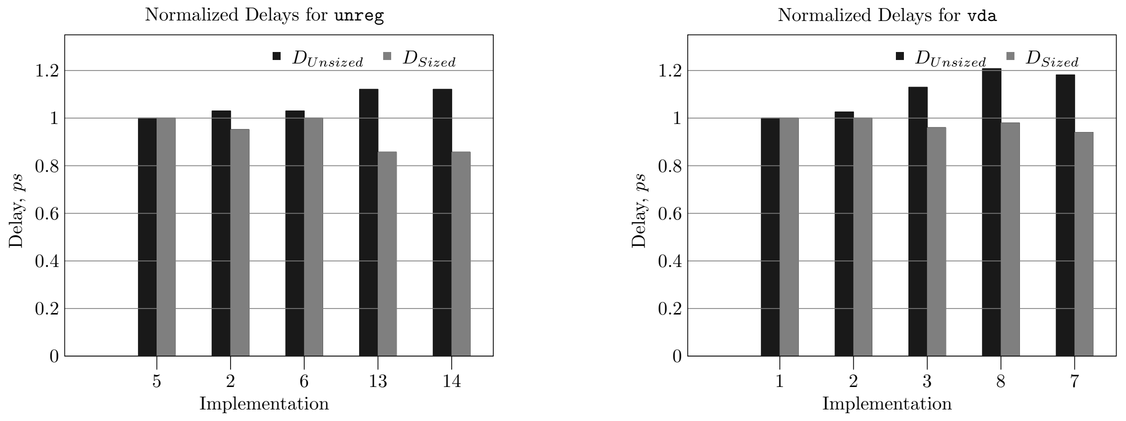


Figure 3.4: Comparing Implementations w.r.t Unsized and Sized Delays

If the reconvergent paths are completely unbalanced, i.e., their structure and logic is such that one always has smaller delay than the other, no errors are introduced due to the manner in which their delay-C_{in} curves are combined. The smallest C_{in} value will consistently be selected for the path with smaller delay. An example of this situation is if the paths correspond to curves k_1 and k_4 in figure 3.3. On the other hand, if the delays of the two paths are roughly of the same order (e.g., if they correspond to curves k_1 and k_2), our approach selects approximately similar values of input capacitances. This may lead to small inaccuracies, since the actual values of input capacitance may be slightly different. However, the error in delay estimation is limited, as shown by the results.

Our approach can also be used to obtain actual sizes of all gates in the circuit. In Algorithm MDE, we can store the value of the load of each output that induces the minimum delay. This information can be used in a forward traversal of the circuit, in order to generate sizes for every gate. A gate with multiple fanins has multiple choices for its size, which can be resolved by selecting the size imposed by the critical input. The effect on the non-critical inputs is that they now have a load different from what was initially assumed. However, the difference in the delays from the primary inputs to the critical and non-critical inputs can be used to compensate for this. In fact, this difference can be usually be used to *reduce* the sizes of the transitive fanin cone of the non-critical inputs, as long as their delay does not become larger than that of the critical input. Gate sizes determined in this manner correspond to a circuit sized for

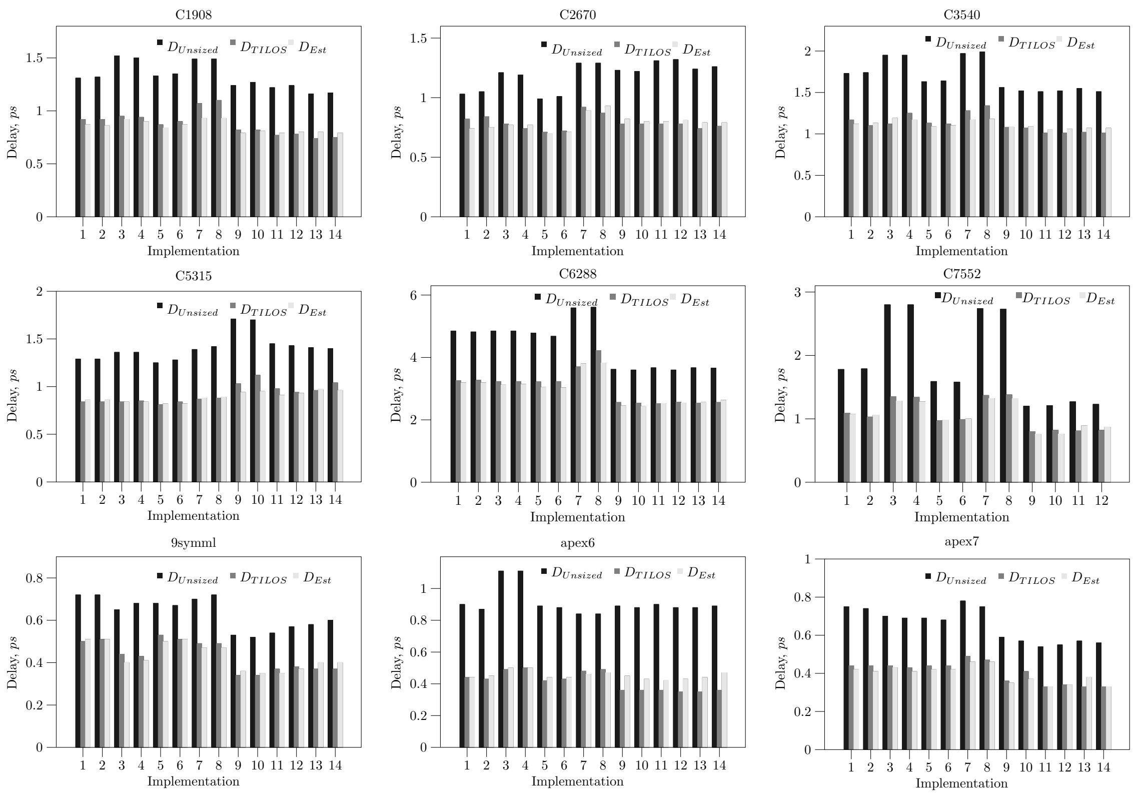


Figure 3.5: Results for Selected ISCAS and MCNC Benchmark Circuits

minimum delay. However, these sizes can also be used as an initial feasible solution for an exact sizing tool, instead of using the original unsized circuit. This can lead to a large improvement in running times of the transistor sizing tool, since a circuit sized using our approach is closer to the final solution than the initial, unsized circuit.

3.3 Results

In order to validate our algorithm, we generated multiple implementations of ISCAS and MCNC combinational benchmark circuits using SIS [SSL$^+$92], and a technology library consisting of minimum sized inverter and two-input NAND, NOR and XOR gates. This choice of gates was selected simply because they have been calibrated in order to obtain accurate values of logical effort and parasitic delay with respect to the models used in our implementation of TILOS [SSH99] describes how these values can

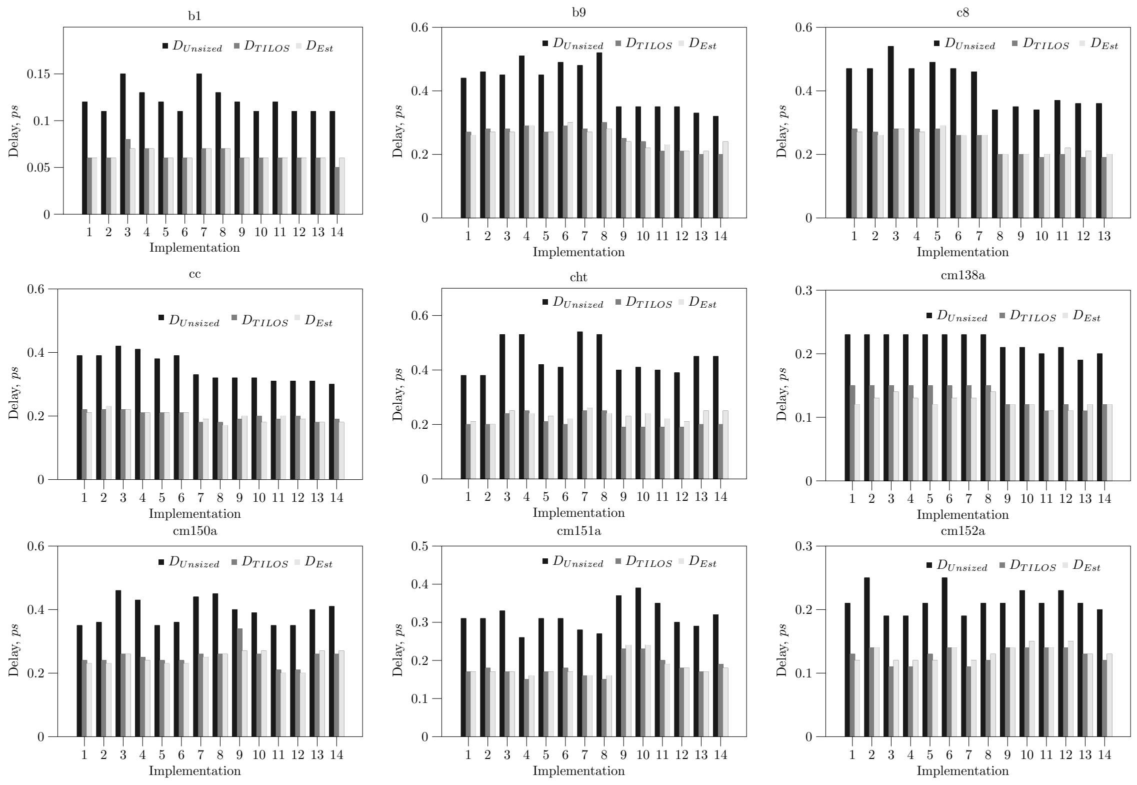


Figure 3.6: Results for Selected MCNC Benchmark Circuits

be obtained from the reference model. In Appendix A, we present a methodology that is used for calibrating circuits using SPICE; we use a similar methodology here, except the calibration is with respect to the models used in our implementation of TILOS. Each benchmark circuit was mapped using different scripts and options, and randomly generated wire parasitics were added to each mapped circuit, in order to simulate the effect of placement and routing considerations. Finally, our implementation of TILOS was used to determine the minimum delay that sizing could realize. This minimum delay was compared with the estimates calculated by Algorithm `MDE`.

Figures 3.5–3.10 presents the comparison of Algorithm `MDE` with TILOS. For each implementation, the first bar represents the delay of the unsized circuit. The second bar is the minimum delay obtained when the mapped circuit is sized using TILOS, and the last bar is the minimum achievable delay estimated using Algorithm `MDE`. As can be seen by the correspondence between the last two bars for each implementation, our

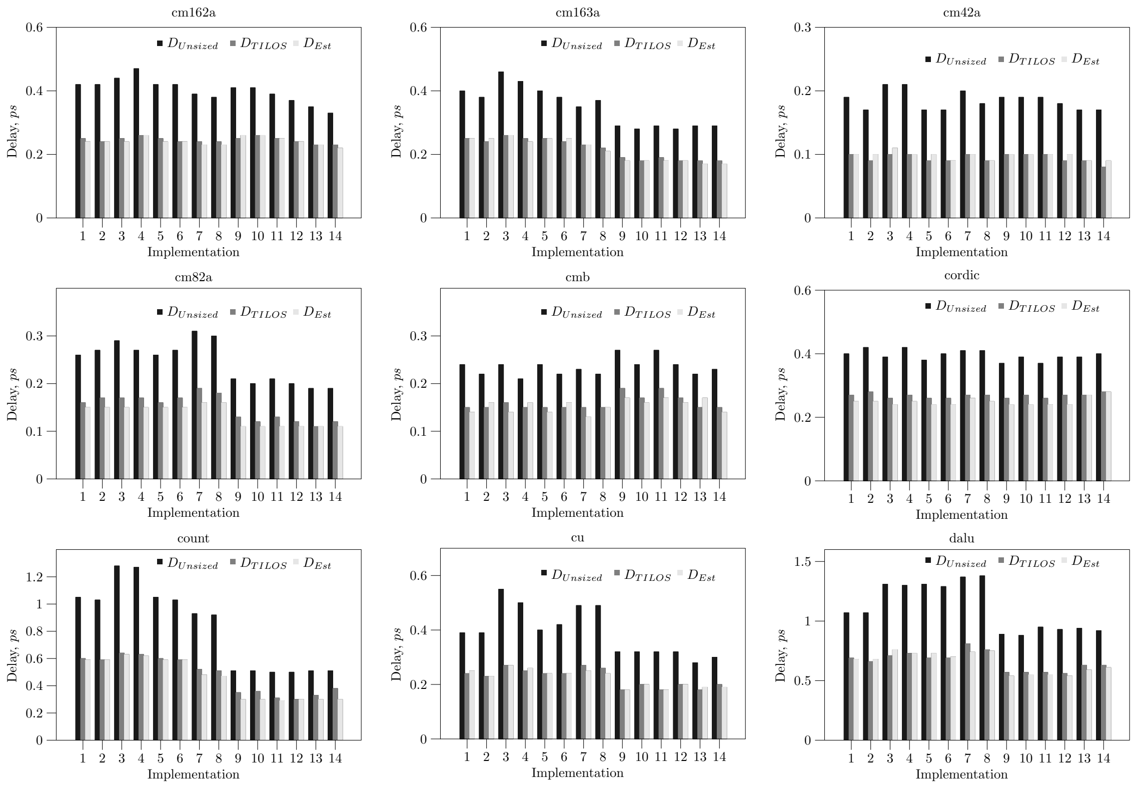


Figure 3.7: Results for Selected MCNC Benchmark Circuits

results agree with those obtained via TILOS. In every case, the execution time for our algorithm was less than a second, while our implementation of TILOS took from a few seconds for C17 up to more than 1500 seconds for C6288. This version of TILOS does not use incremental timing analysis, which would improve its performance drastically. However, given the difference in runtimes of the two approaches, our approach will still be drastically faster than TILOS. The average error between the minimum achievable delay as predicted by Algorithm `MDE` and as obtained using our implementation of TILOS for the circuits shown in Figure 3.5–3.10 is presented in Table 3.1. Including all benchmark circuits (59 in all), the average error is 6.01%.

Another interesting point to note is that comparisons based on unsized circuit delays can be misleading. Consider Figure 3.4, which presents the *normalized* unsized and sized delays of five different implementations of benchmark circuits `unreg` and `vda`. The implementations are ordered in increasing order of unsized delays, repre-

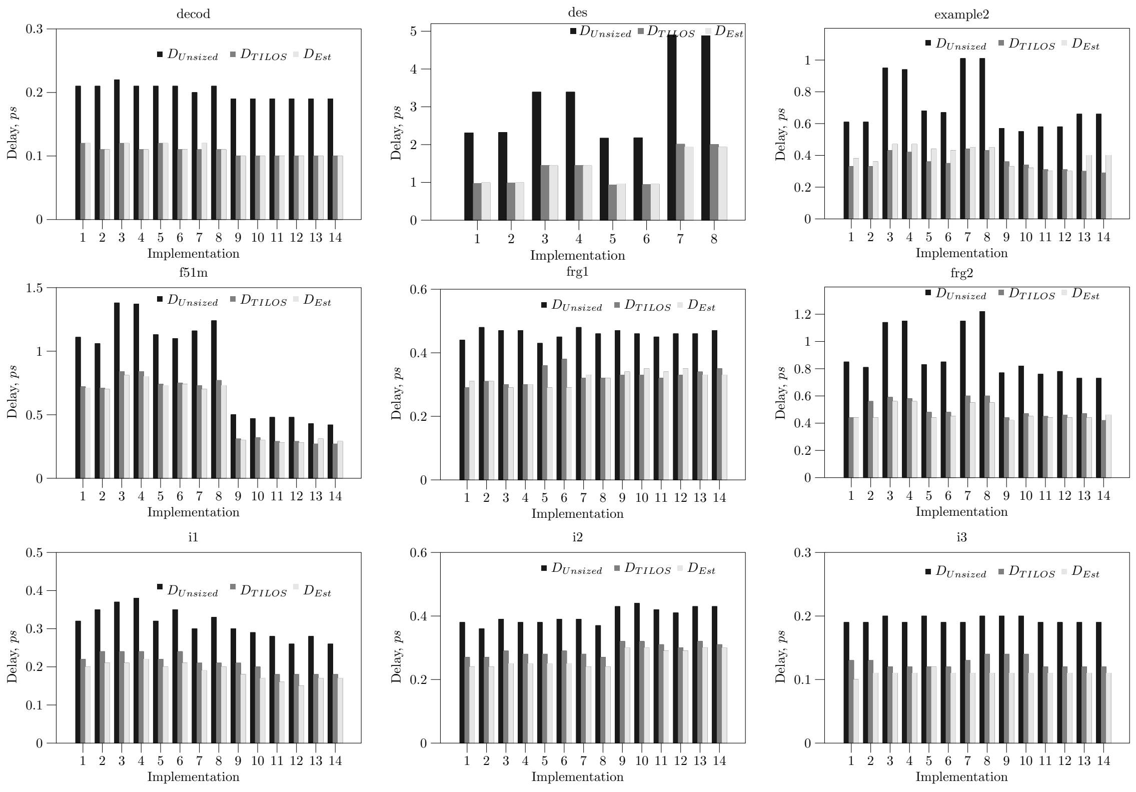


Figure 3.8: Results for Selected MCNC Benchmark Circuits

sented by the first bar. In terms of unsized circuit delays, implementation 5 is the fastest, with implementations 2 and 6 being 3% slower, and implementations 13 and 14 being 12% slower. However, after sizing, we observe drastically different behaviour; implementation 2 is actually 5% faster, implementation 3 achieves the same delay, and implementations 13 and 14 are *15% faster*, as shown by the second bar. Thus, a naive approach to evaluating implementations would have chosen implementation 5, and would have foregone the superior solutions afforded by implementations 13 and 14. This inversion of which implementation is better after sizing, as compared to which is better prior to sizing can be seen in circuit vda as well, where implementation 7 is slower than implementation 1 by 20% in terms of unsized delay, but after sizing, it is 6% faster. In all, comparing multiple implementations of the same circuit shows that this situation occurs in approximately 10% of all cases.

Table 3.1: Percentage Error of `MDE` w.r.t TILOS

Circuit	Error%	Circuit	Error%	Circuit	Error%
C1908	5.50	C2670	4.53	C3540	4.79
C5315	3.76	C6288	3.42	C7552	4.45
9symml	4.47	apex6	11.59	apex7	4.52
b1	2.32	b9	5.10	c8	3.50
cc	3.64	cht	11.74	cm138a	8.86
cm150a	4.73	cm151a	3.10	cm152a	5.26
cm162a	2.05	cm163a	2.75	cm42	3.99
cm82aa	10.16	cmb	8.00	cordic	6.95
count	6.40	cu	2.42	dalu	3.86
decod	0.65	des	1.97	example2	13.53
f51m	4.41	frg1	6.18	frg2	6.61
i1	10.46	i2	9.17	i3	12.60
i10	2.42	lal	4.84	majority	2.68
mux	12.88	myadder	4.57	pair	3.66
parity	9.74	pcle	1.54	pcler8	5.55
pm1	8.17	rot	4.41	tcon	7.49
term1	6.99	unreg	4.06	vda	6.67
x1	6.64	x2	4.19	x3	7.45

3.4 Conclusion and Future Directions

In this chapter, we have presented an algorithm that quickly and accurately estimates the performance improvement that can be obtained in a circuit via transistor sizing. Current placement tools try to provide a solution that is delay-optimal, among other objectives. However, they ignore the gains that may be obtained via sizing. Our approach can be used to guide the placement tool, in effect making it "transistor-sizing aware," so that the final solution is globally optimal.

The approach for optimally distributing load capacitance over multiple fanouts presented in this chapter is a general one, which can potentially be applied to other areas as well. For example, in chapter 5, we integrate it with the technology mapping problem, to obtain optimal gate sizes during technology mapping. Another area of application is in determining the area-delay curve of a circuit, which is presented in the following chapter.

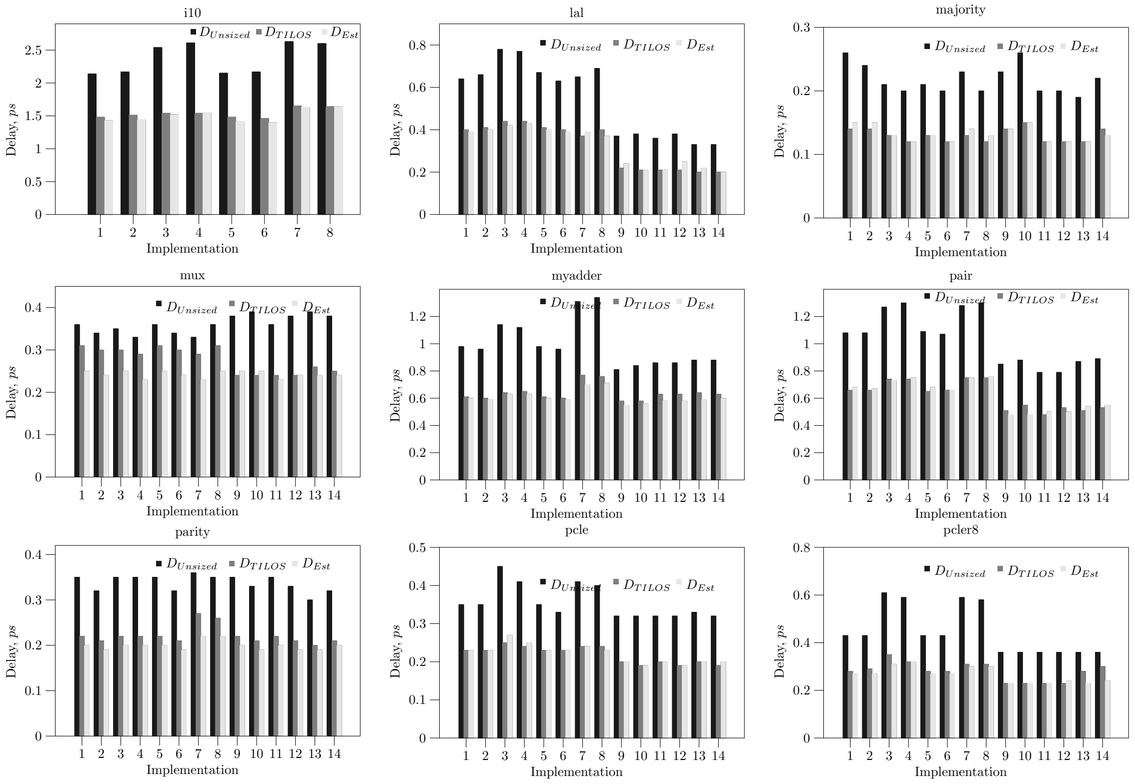


Figure 3.9: Results for Selected MCNC Benchmark Circuits

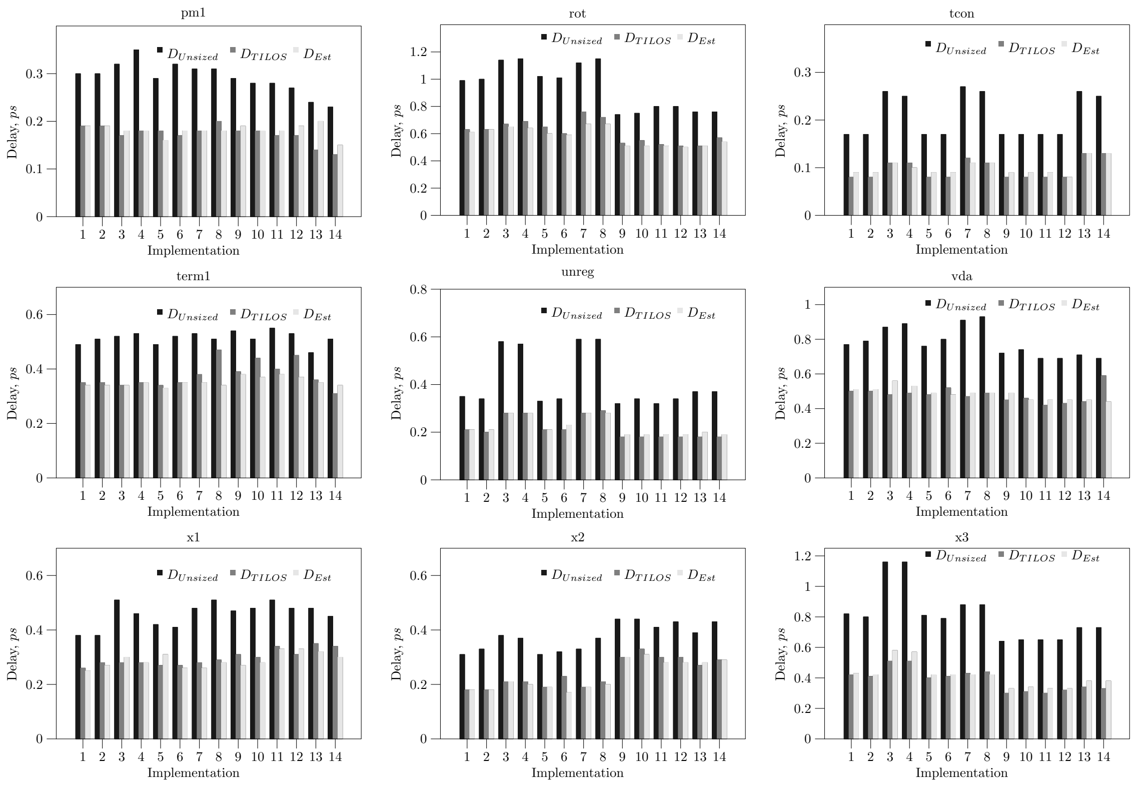


Figure 3.10: Results for Selected MCNC Benchmark Circuits

4 Fast Estimation of Area-Delay Trade-offs using Circuit Sizing

Sizing a circuit can improve performance drastically, as seen from the results in Chapter 3. However, since sizing is a time consuming transform, it is difficult to compare different implementations of a circuit in terms of the cost overhead required for a particular delay target. In this chapter, we extend the minimum-delay estimator presented previously to a fast estimator of the complete area-delay trade-off curve of a given circuit, allowing a designer to choose the most appropriate implementation for a given delay. We observe excellent fidelity with the actual area-delay curves (98.94% correct comparisons), with an average error of 5.76% in the area differences predicted.

4.1 Introduction

After a circuit has been placed and routed, it can be sized in order to improve performance, incurring cost overheads, which could be area or power. As mentioned in the previous chapter, the current transistor sizing tools ([FD85,SRVK93,CCW98,SSP02, CCH+98, BVSH02]) have large running times – sizing a reasonably large circuit can take up to a few hours. In Chapter 3, we considered the case of sizing an implementation to the minimum delay. We now consider the situation where we have a target delay, which may not be the smallest achievable delay. As before, if a designer is presented with a number of implementations of the same functionality, he would prefer selecting only the best for further sizing. This leads to the question of which is the best implementation, i.e., which implementation will incur the lowest cost, when sized to meet a particular delay. For convenience, we use the *area* of the implementation as a measure of the cost. There is a direct correlation of area with other measures of cost, such as power dissipation, sub-threshold leakage and gate leakage, and a similar approach can be used when the cost function is power, or a weighted combination of area and power. Answering the question of determining the lowest cost (area) implementation requires knowledge of the area-delay curve of each implementation, but these are determined only after sizing has been carried out. We therefore present an approach that *estimates* the area-delay curve of a given implementation. We do not use a sizing tool, and therefore, our approach is fast, and as we will show, the estimated curve has high fidelity with the actual area-delay curve.

Chapter 3 presents an approach based on of logical effort [SS91, SSH99], for determining the minimum achievable delay of an implementation, if transistor sizing were to be applied to it. While this is a useful metric to have, it is not sufficient for comparing circuits that will be sized to arbitrary (non-minimum) delay points. This drawback is illustrated in the following section, where we present the importance of determining the entire area-delay curve of an implementation. We then show how the area-delay curve can be estimated by using the information stored in the minimum achievable delay calculation. Finally, we apply the approach presented here to comparing different implementations of the same benchmark circuits, and show how accurate comparisons can be made quickly.

4.2 Problem Formulation

Figure 4.1(a) shows the area-delay curves of multiple implementations of benchmark circuit C7552. Each implementation was obtained by varying parameters given to the optimization and synthesis tool. The area-delay curves were obtained using our implementation of TILOS [FD85]. In these plots, the area of an implementation is shown on the y-axis, and delay on the x-axis. The extreme right point of each curve corresponds to the unsized circuit; this has maximum delay and the smallest area, and successively smaller delay values require larger areas. Note that the curves have a characteristic point (called the 'knee'), at which the rate of change of area with respect to delay changes drastically.

Each curve is bounded by the maximum delay (i.e., the unsized circuit delay)

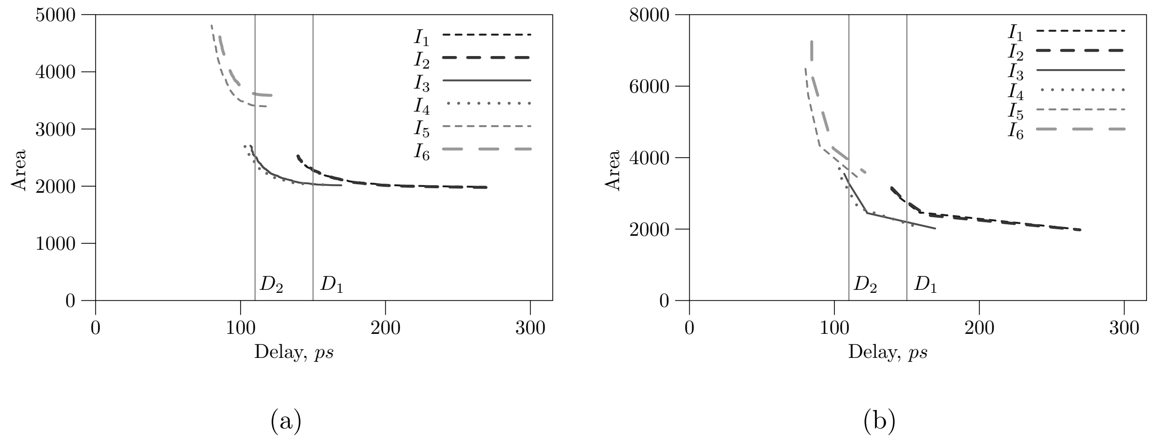


Figure 4.1: Area-Delay Curves of 6 Implementations of Benchmark Circuit C7552 (a) Generated using TILOS and (b) Estimated

and the minimum achievable delay. However, as can be seen, the *shape* of each curve can vary significantly. For example, in the curves shown in Figure 4.1(a), the knee of each curve can either be closer to one of the end points or in the center. This property varies between different circuits, as can be expected, but it also varies between implementations of the same circuit. For implementations I_1 and I_2 of C7552, the knee is closer to the minimum delay point. Hence, we initially observe large improvements in delay for relatively small area cost, for these implementations, but further delay improvement comes at the cost of large increases in area. The situation is reversed for implementations I_5 and I_6, where the knee is closer to the maximum delay point. In this scenario, trying to determine which implementation is the best at some intermediate delay point without having knowledge of the entire area-delay curve is difficult.

Suppose a designer wants to determine the best implementation among those available for some target delay of D_1. Calculating the minimum achievable delay and the unsized circuit delay of all implementations, the designer can determine that implementations I_1, I_2, I_3 and I_4 meet this target delay. At a different target delay of D_2, the implementations that have to be considered are I_3, I_4, I_5 and I_6. Implementations I_1 and I_2 need not be considered, since their minimum achievable delay is larger than this value. However, this information is not sufficient, since *which* of these circuits should be selected is still not known. Ideally, he would like an ordering of these implementations based on the cost, which in this case, is the area. The required ordering for a delay of D_1 is $\{I_3,\ I_4,\ I_2,\ I_1\}$, and for D_2 it is $\{I_4,\ I_3,\ I_5,\ I_6\}$. Simply ranking implementations based on the unsized delays and areas is not enough, e.g., at one delay point, I_4 has lower area, and at the other I_3 is better. This situation, of different implementations being the best at different delay points, is also seen in implementations of other benchmark circuits.

Recall that using a sizing tool to obtain the area-delay curves of *one* implementation of a circuit is time-consuming. Obtaining the area-delay curves of multiple implementations is prohibitively expensive. Our heuristic, presented in the following section, addresses this issue by *estimating* the area-delay curve of a given implementation. These curves can be used to compare different implementations in two ways. First, given a target delay, we can generate a cost-based ordering based on the estimated area-delay curve of each implementation. Second, instead of calculating the actual areas at the delay value of interest, we measure the relative area difference between the implementations. The relative area difference has a good correlation with the actual area difference, and can be estimated quickly.

The area-delay curves obtained using our approach are as shown in Figure 4.1(b). A rough comparison with the plots of Figure 4.1(a) shows that this heuristic captures the behavior of the area-delay curves well. In particular, the shape of the estimated curve, with respect to the position of the knee matches that seen in the actual area-delay curves. A complete comparison is made in section 4.4.

4.3 Area-Delay Curve Estimation

Our starting point is Algorithm `MDE`, presented in chapter 3 and [KS04]. It estimates the minimum achievable delay of a circuit, by calculating the Delay-C_{in} curve for each gate of the circuit. This curve stores the best possible delay from the input of a gate to any primary output, for each input capacitance value, corresponding to the gate size. It also implicitly stores the sizes of gates on the path to the primary output, which achieve this delay. Determining the Delay-C_{in} curve of a gate that has a single fanout is relatively straightforward, since using dynamic programming, the

Delay-C_{in} curve of the immediate output is the only one that has to be considered. For gates with multiple fanouts, all points on the Delay-C_{in} curves of each fanout have to be considered, and the the number of possible combinations of these points can be extremely large. However, Delay-C_{in} curves of multiple fanouts can be combined in a manner that leads to tractable run times without any loss of information. The Delay-C_{in} curve of a gate thus captures the delay characteristics of the entire transitive fanout cone of the gate in a compact and elegant formulation. Once the Delay-C_{in} curves at the primary inputs have been calculated, the minimum achievable delay of the circuit can be determined by selecting the minimum delay point from these curves. The gate sizes associated with the selected point can be propagated to the primary outputs, and adding these sizes gives us an estimate of the area required to meet the selected (minimum) delay.

Since we are interested in determining the area-delay curve of the implementation, the obvious approach is to calculate the area with the delays during the Delay-C_{in} calculation. However, there are a few problems with this approach. There are multiple configurations of gate sizes that can achieve the same delay value, and hence multiple solutions for each delay value have to be stored. These enhanced Delay-C_{in} curves do not have the optimal substructure property, and hence we can no longer use dynamic programming. Finally, every combination of points in the enhanced Delay-C_{in} curves of multiple fanouts has to be considered, which further increases the complexity.

We therefore need another approach to estimating the area-delay curve. Recall that the Delay-C_{in} curves calculated in Algorithm `MDE` implicitly store sizes of gates in the transitive fanout cone required for for achieving the minimum delay for each value of C_{in}. Hence, we can size the circuit using points on the Delay-C_{in} curves of the primary inputs, and calculate the corresponding area. However, these points may not be optimal i.e., the area calculated using the above approach may not be the smallest area for a particular delay. For example, say we have a minimum delay of d_1 for C_{in_1} and d_2 for C_{in_2}, with corresponding circuit areas of a_1 and a_2, and $d_1 > d_2$. It is possible that there was a non-minimum delay $d'_2 = d_1$ for an input capacitance of C_{in_2} that had a corresponding circuit area a'_2, that is less than a_1. The solution (a'_2, d'_2) is clearly better than the (a_1, d_1) solution, but since only minimum delay points are considered, the superior solution is hidden.

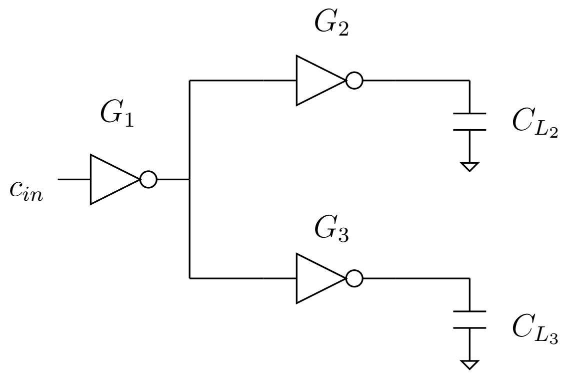


Figure 4.2: Example Circuit

Consider the circuit shown in Figure 4.2, with two branches of the circuit driving different loads [1]. For some input capacitance of c_{in}, we obtain a number of delay values, the minimum of which is stored in the Delay-C_{in} curve, and the other delay values are discarded. However, we can size the circuit using the minimum as well as the discarded delay values (this is for the same input capacitance of c_{in}), and calculate the corresponding areas. These points are shown in Figure 4.3, and the best points for an area-delay curve perspective are the ones marked by a line. This procedure can be repeated for other values of C_{in}, and the union of the solutions obtained gives us the area-delay curve desired. This is shown in Figure 4.4 for three values of C_{in}. Note the intersection in the curves corresponding to $c_{in} = 4$ and $c_{in} = 5$, this is an example of sub-optimality if only the minimum delay points were to be considered.

Thus, we estimate the area-delay curve of a circuit by sizing it for different values of delay, for every value of C_{in} and measuring the area. In order to keep the run time low, rather than sizing for all delay values, we size the circuit for a limited number of values (in our experiments, we found that selecting 10 sub-optimal delay points was sufficient). This has an impact on the accuracy of our results, but the effect is limited, especially since our focus is on comparing implementations, rather than on determining actual areas.

Our heuristic, called Algorithm `ADC` (shown in Algorithm 4.1) is obtained by modifying Algorithm `MDE` as follows. At the primary inputs, we store *sets* of Delay-C_{in} curves. Each time $D_{\mathrm{G}\rightarrow PO}[c_i]$ is updated to a new value, we store the replaced value

[1]We use inverters for simplicity of presentation; this discussion extends to other gates as well.

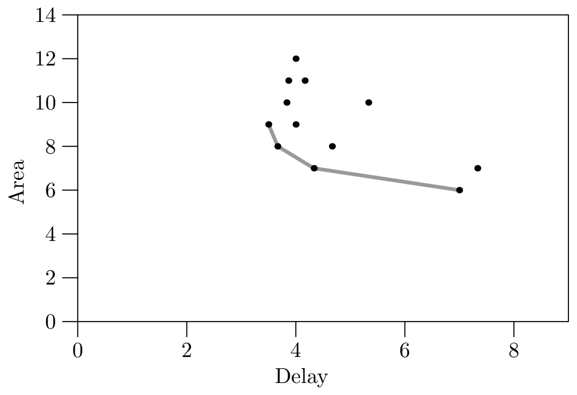


Figure 4.3: Calculating the Area-Delay Curve for one value of c_{in}

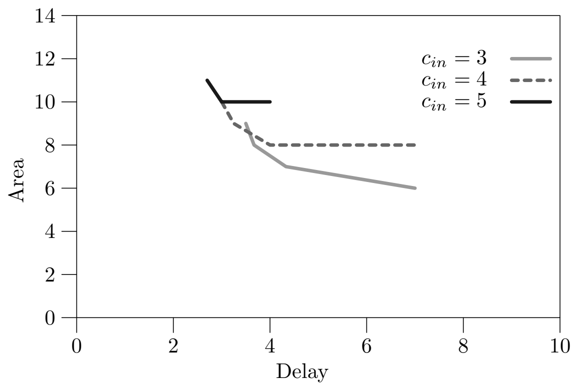


Figure 4.4: Area-Delay Curve of the Circuit in Figure 4.2

as an entry in secondary curves. The minimum delay value from these secondary curves are then used to size the circuit, and obtain other points on the delay-area curve. Circuits sized in this manner have greater delay than the minimum achievable delay, and after area recovery, they have smaller area as well.

The solution obtained using this approach is naturally not exact. However, as discussed above, since the auxiliary data of points on the secondary curve encode sizes of the outputs (and particularly, of sizes of multiple fanouts), these solutions still provide a good representation of the area behavior of the circuit at different delay points. i.e., though we cannot use the area-delay curves to make absolute judgments, we can still make comparative judgments between different circuits.

Once the circuit has been sized, we determine the arrival and required times at

Algorithm 4.1 `ADC: Area-Delay Curve Calculation`

```
for each gate G whose outputs have been processed do
  if G is not a PI then
    Calculate Delay-$C_{in}$ curve of G as in Algorithm 3.1
  else
    Calculate Delay-$C_{in}$ curve as before, but for each $c_i$ store *all* solutions
  end if
end for

for each set of Delay-$C_{in}$ curves of the PIs do
  Minimum Delay = $\max\{\min_{\text{all PI's}}\{\text{delay to PO}\}\}$
  // forward traversal
  Size the circuit based on the selected point. Also determine the arrival time at
  each gate
  // reverse traversal
  Determine the required time at each gate
  // area recovery
  for each gate G in reverse topological order do
    $slack$ = arrival time − required time
    while $slack > 0$ do
      reduce the size of G
      update the arrival and required times of G and its inputs
    end while
  end for
  Determine area and delay of the sized circuit
end for
```

each gate, and use the slack to reduce the sizes of the gates. This step can drastically reduce the area of a circuit, since the non-critical parts of the circuit are usually sized to be unnecessarily fast.

Algorithm 4.1 is almost as fast as Algorithm 3.1. Once the Delay-C_{in} curves have been calculated, the actual calculation of arrival and required times only needs two traversals of the circuit, and sizing each gate requires a maximum of $O(m)$ operations, if there are m gate sizes available. This is done a fixed number of times, for each set of Delay-C_{in} curves that have been calculated. Thus, the running time is dominated by that of running Algorithm 3.1.

Table 4.1: Full ADC Comparison

Circuit	Comparisons		E_{Total}		E_{False}	
	Total	False	max.(%)	avg.(%)	max.(%)	avg.(%)
C432	102	3	21.85	6.42	11.33	9.51
C499	158	14	21.52	6.38	16.08	7.83
C880	41	3	13.95	4.11	9.89	6.93
C1355	136	10	28.61	8.17	18.11	8.02
C1908	113	4	25.72	5.62	5.22	4.00
C2670	121	1	18.87	3.49	3.28	3.28
C3540	101	5	18.24	4.49	14.51	8.12
C5315	163	8	27.51	7.24	5.09	2.34
C6288	57	10	22.50	4.65	11.62	4.85
C7552	30	2	25.08	7.02	4.70	2.68
Total	1022	60(5.87%)				
Max.			28.61		18.11	
Avg.				5.76		5.75

4.4 Results

In order to validate our algorithm, we generated multiple implementations of the ISCAS combinational benchmark circuits using SIS [SSL$^+$92], and a technology library consisting of minimum sized inverter and two-input NAND, NOR and XOR gates. These gates are calibrated to obtain accurate values of logical effort and parasitic delay with respect to the models used in our implementation of TILOS. Each benchmark circuit was mapped using different scripts and options, and randomly generated wire parasitics were added to each mapped circuit, in order to simulate the effect of placement and routing considerations. Finally, our implementation of TILOS was used to determine the actual area-delay curves, against which the estimated curves obtained by Algorithm `ADC` can be benchmarked.

The first goal of our approach is to correctly predict which implementation is the best for different delay points. Our methodology for measuring the effectiveness of Algorithm `ADC` is as follows. For the entire range of possible delay values, we select ten equally spaced delay points. Note that the number of implementations that can be sized to meet a particular delay value varies by circuit. We make pairwise comparisons between all implementations available at the selected delay point, and determine which implementation is better. In Table 4.1, for each benchmark circuit, the number of comparisons made are shown in the second column. Next, we make the same comparison using the delay curves obtained from our implementation of TILOS. An incorrect comparison is when the ranking according to Algorithm `ADC` is different from that obtained from TILOS. As shown in the next column, incorrect comparisons occur only 5.87% of the time.

Next, we measure the error in the predicted area difference. Let implementations I_1 and I_2 have estimated areas of $A_{I_1\text{est}}$ and $A_{I_2\text{est}}$, and assume $A_{I_1\text{est}} < A_{I_2\text{est}}$, so that I_1 is the better implementation. The difference between the estimated areas of I_1 and I_2, is calculated as $\Delta A_{\text{est}} = 100(1 - \frac{A_{I_1\text{est}}}{A_{I_2\text{est}}})$. Similarly, the difference between the areas from the actual area-delay curves, $A_{I_1\text{act}}$ and $A_{I_2\text{act}}$ is calculated as $\Delta A_{\text{act}} = 100(1 - \frac{A_{I_1\text{act}}}{A_{I_2\text{act}}})$. The absolute error of our approach is $E = |\Delta A_{\text{est}} - \Delta A_{\text{act}}|$, and the maximum average value of this error over all comparisons are presented in columns 4 and 5 of Table 4.1. The maximum error is high, but it does not happen often, and over all circuits, the average error is 5.76%. The last two columns present the maximum and average errors in area estimation for comparisons that were mis-predicted. Once again, while the maximum is large, it is rare, and the average error in this case is 5.75%.

5 Library Based Technology Mapping using Logical Effort

In this chapter, we move up the design flow presented in Figure 1.1 in Chapter 1. Thus far, we have focused on *estimating* the changes that can be obtained by the sizing transform, in terms of minimum delay and the cost overhead for a given target delay. We now combine our solution to the load distribution problem with the mapping transform, thereby obtaining superior solutions. We also propose a new approach to library-based technology mapping, which is also based on the method of logical effort. Our algorithm is close to optimal for fanout-free circuits, and is extended to solve the load-distribution problem for circuits with fanout. On average, benchmark circuits mapped using our approach are 32.48% faster than the solutions obtained from SIS.

5.1 Introduction

The logic synthesis portion of design implementation consists of technology independent optimization, followed by technology mapping. A number of algorithms have been proposed for the latter step, such as tree-mapping [Keu87] and DAG-mapping [KBS98], using load-dependent delay models [TMBW90], constant delay models [GLH^{+}95,SIS99] as well as using logical effort [HWKMS03]. High-performance designs require rich libraries, with multiple instances of each cell, which have varying delay, area and drive capabilities. Technology mapping, therefore, is not simply identifying the best cells to be used to implement some logic, but also the best instance of the selected cells.

Traditionally, logical effort has been used as a quick means of estimating the delay of a path of logic. Given load and input capacitances, it can also be used to find the minimum delay that can be achieved by a path, and the corresponding gate sizes that lead to this minimum delay. Thus, it can be applied to making comparisons of different implementations of the same functionality for a path, but falls short when comparing entire circuits. In this work, we apply logical effort to the problem of minimum-delay technology mapping. Our approach for selecting matches during technology mapping has a couple of advantages over previous methods. Firstly, size selection of gates in the solution is implicit in our formulation, and does not have to be determined during matching. Secondly, the delay model is inherently load-dependent, and there is no need to enumerate solutions for all possible load values, as is traditionally done [TMBW90]. This makes our approach faster than current algorithms for fanout-free circuits.

We also modify the previous solution to the load distribution problem, to accurately handle multiple fanouts. In Chapter 3, this problem was addressed in the context of sizing a mapped circuit. We use the approach presented there to guide the technology mapping algorithm at multiple fanout points in the circuit, leading to mapped circuits that have better performance than solutions obtained by previous methods.

5.2 Traditional Technology Mapping

In this section, we briefly summarize the state of technology mapping and the drawbacks of current mapping algorithms.

Cell- or library-based technology mapping is the process of binding a technology independent logic level description of a circuit to a library of gates in the target technology. A dynamic-programming algorithm based on tree covering was proposed in [Keu87], and has served as the basis of later technology mapping algorithms. This is a two-step algorithm –

- In the *matching* step, matches for all gates are generated in an input-to-output traversal of the circuit, and the optimum match (based on its cost and the cost at its inputs), and the corresponding matches at the inputs, is stored as the solution for that gate.

- In the *covering* step, the solution for the entire circuit is generated by an output-to-input traversal of the circuit. At the primary outputs, the best match is selected, and the covering recurses on the inputs of this match.

One of the drawbacks of this approach is that the circuit to be mapped (the "subject graph") is partitioned into disjoint fanout-free trees, which are then optimally mapped. However, this leads to restrictions on the solutions, since matches cannot cross tree boundaries. In [KBS98], it was pointed out that if duplication at tree boundaries were to be allowed, DAG-mapping, as opposed to tree-mapping, would provide superior results. However, [KBS98] does not address the load-distribution problem, described shortly.

The delay models used in technology mapping fall into the following categories –

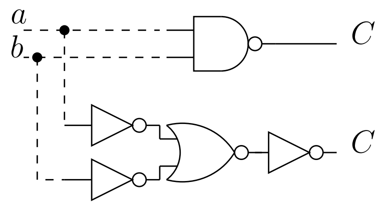
Load-Independent Delay Models assume that the delay of a cell does not depend on the load being driven, which is unrealistic. However, during technology mapping (even in the case of fanout-free regions), the load is not known until the covering step, and assuming load-independence of delay is convenient. Naturally, this model is not widely used.

Load-Dependent Delay Models express delay as a polynomial function of the load being driven. Higher-order delay functions, such as quadratic functions, can be used for greater accuracy, although linear functions, as used in [TMBW90], also suffice. Technology mapping using such a delay function generates optimal matches for all load values in the matching step. During covering, the actual value of the load is known, and the corresponding match can be selected as the solution.

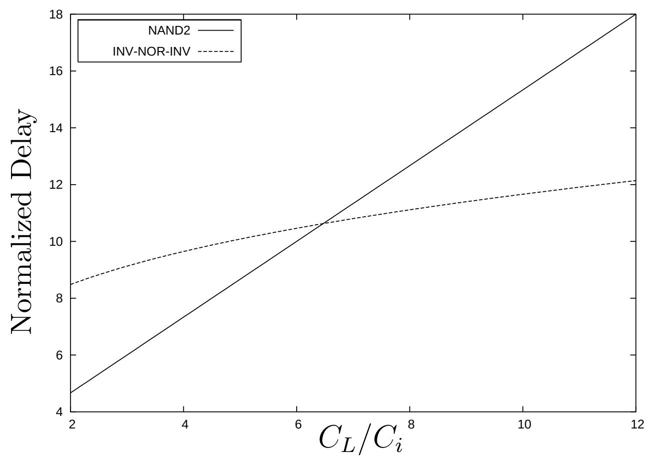
Constant Delay Models assign a fixed delay to each library cell (note that this is not the same as the load-independent delay models). Technology mapping is carried out under the assumption that given a load, these cells can be sized in order to achieve the assigned delay. These models have been used in [GLH$^+$95, SIS99], but the main drawback is that selecting cells during matching is sensitive to the load being driven.

Gain-Based Delay Models express the delay of a gate as a function of the ratio of output-to-input capacitance of the gate, and have been applied to technology mapping in [HWKMS03]. However, the selection of sizes of gates in [HWKMS03] is based on an ill-defined parameter called *global gain*, whose value is set either by experimentation or relies on the intuition of the designer.

Load-Dependence of Optimal Matches



(a) Example Circuit



(b) Delay Vs. $\frac{C_L}{C_{in}}$

Figure 5.1: Influence of Load on Solutions

Consider Figure 5.1(a), where output C is the NAND of two inputs, a and b. This functionality can be obtained by either selecting a NAND2 gate directly, as shown on the top, or by selecting an INV-NOR2-INV chain as shown at the bottom. It may seem that the smaller solution will outperform the larger one. However, consider the delay equations for each option, assuming the following values: $g_{\text{INV}} = 1$, $g_{\text{NAND2}} = \frac{4}{3}$, $g_{\text{NOR2}} = \frac{5}{3}$, $p_{I\text{NV}} = 1$ and $p_{N\text{AND2}} = p_{\text{NOR2}} = 2$. The minimum delay that can be achieved by each option can be calculated using Equation (2.7) of Chapter 2, as

$$\begin{aligned}\hat{D}_{\text{NAND2}} =& \frac{4}{3} \times \frac{C_L}{C_i} + 2 \\ \hat{D}_{\text{INV-NOR2-INV}} =& 3 \cdot \left[\frac{5}{3} \times \frac{C_L}{C_i}\right]^{\frac{1}{3}} + 4 \end{aligned} \tag{5.1}$$

Figure 5.1(b) plots the minimum delay of Equation (5.1) as a function of the electrical effort, $\frac{C_L}{C_i}$. It is obvious that there is no universally better choice in this case – for small loads, the NAND2 has lower delay, while the INV-NOR2-INV is better for larger loads.

This issue arises in traditional technology mapping in two contexts. Firstly, we can see the drawback of constant delay models, where the match generation is independent of the load. Secondly, at multiple fanout points, approaches using load-dependent delay models can only *estimate* the load being driven, and make the appropriate selection. If this estimate is wrong, suboptimal solutions are generated.

We address this issue by proposing a new approach to technology mapping in the next section. This approach is optimal for fanout-free structures. We then show how our algorithm can be combined with a modified version of the solution to the load distribution problem, in order to obtain mapped solutions that are better than those obtained by traditional methods.

5.3 Logical Effort Based Technology Mapping

In this section, we show how we extend logical effort in order to map a circuit to a target library. We first show how fanout-free circuits can be mapped using logical effort, followed by our approach for multiple fanouts, where we provide a solution to the load-distribution problem. We finally summarize our overall approach for general circuits.

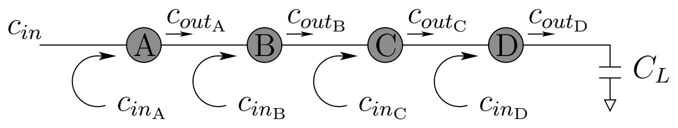
5.3.1 Mapping Fanout-Free Circuits

In the matching step of the traditional technology mapping algorithm, all possible matches are generated at each gate in the subject graph, and the best match is stored as a potential solution. For minimum-delay mapping, the best match is determined on the basis of its delay, plus the maximum delay at its inputs. We use a similar approach, but rather than delay, we minimize the cumulative path logical effort, G. For ease of explanation, we first consider a simple path of logic (with each gate having single fanins), and then present our approach for multiple fanins.

Consider Equation (2.7), which is reproduced below

$$\hat{D} = N\,(GH)^{\frac{1}{N}} + P \tag{5.2}$$

As mentioned before, the path electrical effort, H, is the product of the electrical efforts of gates on the path. However, this product telescopes as shown in Figure 5.2, since the input capacitance of a gate is the load capacitance of its input (e.g., $c_{in_\mathrm{C}} = c_{out_\mathrm{B}}$). Hence, H can be calculated as $\frac{C_L}{c_{in}}$, where C_L is the load being driven by the last gate, and c_{in} is the input capacitance of the first gate on the path under consideration. Thus, if the electrical effort of a path is known, its delay can be calculated using Equation (5.2), *without knowing the sizes of each gate on the path.* We show later how the individual gate sizes can be calculated.



$$\begin{aligned} H &= \frac{c_{out_\mathrm{A}}}{c_{in_\mathrm{A}}} \times \frac{c_{out_\mathrm{B}}}{c_{in_\mathrm{B}}} \times \frac{c_{out_\mathrm{C}}}{c_{in_\mathrm{C}}} \times \frac{c_{out_\mathrm{D}}}{c_{in_\mathrm{D}}} \\ &= \frac{C_L}{C_{in}} \end{aligned}$$

Figure 5.2: Calculating the Electrical Effort of a Path

During technology mapping, we have the freedom of choosing which gates are used to implement the required logic. If we temporarily assume that the path has a fixed number of stages (this will be relaxed shortly), then for a given path electrical effort, we can see from Equation (5.2), that the minimum delay over all possible implementations is obtained by the implementation that minimizes the path logical effort, G.

Next, we allow any number of stages for the implementation, and keep track of the optimal solution for all path lengths of the matches. In this case, at the primary output, we obtain a set of solutions, each of which implement the logic using a different path length. We can use Equation (5.2) to determine which of these gives us the minimal delay.

Once the values of G, H and N that minimize the delay have been determined, we can calculate the sizes of each gate on the path. As mentioned previously, the minimum delay is achieved when each stage in the path bears the same amount of effort, and this effort $\hat{f} = F^{\frac{1}{N}}$ [SSH99]. Recall that the stage effort f_i of stage i is the product of its gate logical and electrical efforts. We use this property to determine the sizes of the gates as follows.

$$f_i \;=\; g_i \cdot h_i \;=\; g_i \cdot \frac{c_{out_i}}{c_{in_i}} \;\Rightarrow\; c_{in_i} \;=\; \frac{g_i}{f_i} \cdot c_{out_i} \tag{5.3}$$

Thus, starting from the primary output, where the load C_L is known, a traversal towards the primary input successively determines the input capacitance (and hence size) of each gate.

Logical effort based technology mapping for a path can therefore be summarized as follows.

- In the matching step, traverse the path from primary input to primary output. For each match at a gate, the cost is the product of the logical effort of the match and the cost at the input of the match. The length of the path is the length of the input of the match plus 1. For all path lengths, store the best match.

- At the primary output, determine the combination of G, H and N that minimize the delay.

- In the covering step, traverse the path from primary output to primary input, generating the solution as in regular technology mapping. Calculate the correct sizes of each gate using Equation (5.3).

We can now generalize this approach to circuits with gates having multiple fanins. Recall that the minimum delay is achieved by minimizing the cumulative logical effort. For a gate t with r inputs $\mathcal{I}_1, \mathcal{I}_2, \dots \mathcal{I}_r$, we define the critical input $\mathcal{I}_c$ to be the input having the maximum delay from a primary input. Consider the situation where t has some input capacitance c_{in_t}, and the path length from primary inputs to any input of t is the same. The delay from the primary inputs to each input of t can be determined using Equation (5.2), with c_{in_t} assuming the role of the load. It is now obvious that the critical input, $\mathcal{I}_c$ (the input with maximum delay) will also be the input with maximum cumulative path effort.

Logical effort based technology mapping for fanout-free circuits can therefore be carried out in a manner similar to the approach for simple paths. The cost of each match is now the product of the logical effort of the match, and the maximum of the costs of its inputs. As before, the delay depends on the length of the path, N. We therefore record solutions for all values of path length at each gate, and at the primary outputs, the best delay over all N can be selected, and the corresponding solution recovered. As we will show in Section 5.4, the average path length N for typical benchmark circuits is between 2 and 5.

The pseudo-code of our dynamic-programming based approach is presented in Algorithm 5.1, and we use Figure 5.3 to illustrate it. Here, a simple chain of three gates, A, B and C is to be mapped to a library of three cells, X, Y and Z, with logical efforts g_{X}, g_{Y} and g_{Z}.

For all legal values of lengths, each gate t keeps track of the accumulated product of logical efforts G_t, and the corresponding matches $\mathcal{M}_t$. G_t and $\mathcal{M}_t$ are indexed according to the length of the path at the inputs to the match at gate t (plus 1 for the match at t itself). Assume that we are considering the match of a library pattern m at gate t, which has logical effort g_m and parasitic delay p_m, and that the length of the path from the primary input to t is n. The cumulative logical effort of length n at input i of the match is $G_i[n]$. We select the maximum of this value over all inputs, and take its product with g_m, to obtain the cumulative logical effort at the output of t for a path of length $n+1$. Finally, the match corresponding to a selected $G_t[n+1]$, and the cumulative parasitic delay $P_t[n+1]$ is also stored.

Algorithm 5.1 `LE-Based Matching for Fanout-Free Regions`

```
// initialize
for each primary input $p$ do
  $G_p[0] = 1$
end for

// Phase I: Matching
for each gate $t$ in topological order do
  // $\mathcal{M}_t$ is the set of all matches at $t$
  set $\mathcal{M}_t[i] = 0$ for all $i$
  for each $m \in \mathcal{M}$, with logical effort $g_m$ do
    // $\mathcal{I}$ is the set of inputs to $m$
    // calculate cumulative effort $G_t[n+1]$ from
    // the inputs, corresponding to distance $n$
    $temp = g_m \times max_{i \in \mathcal{I}} G_i[n]$
    if $\mathcal{M}_t[n+1] = 0$ OR $temp < G_t[n+1]$ then
      $G_t[n+1] = temp$
      $P_t[n+1] = p_m + P_i[n]$
      $\mathcal{M}_t[n+1] = m$
    end if
  end for
end for

// Phase II: Selecting Solution
at the primary output, select the combination of $G$, $H$ and $N$ that minimizes delay

// Phase III: Covering
select matches in a traversal from the primary output to primary inputs, sizing the
matches appropriately
```

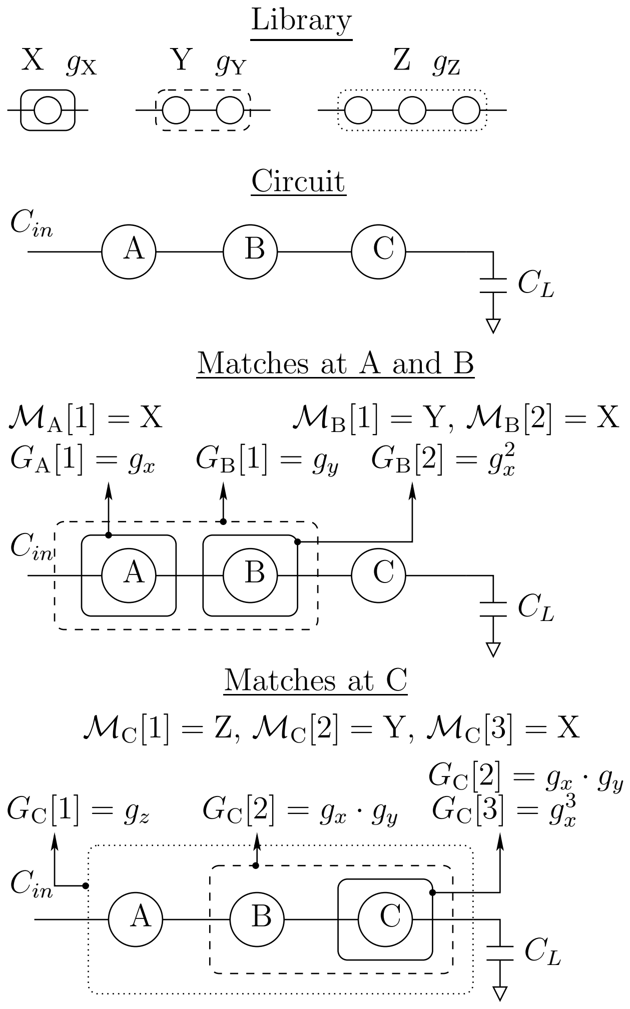


Figure 5.3: Example of LE-Based Technology Mapping

In the example, the only match of a library pattern at gate A is that of pattern X, and the corresponding solution for A is $G_A[1] = g_x$, $\mathcal{M}_A[1] = X$. At gate B, however, we have two possible matches, the match of X, with solution $G_B[2] = g_x^2$, $\mathcal{M}_B[2] = X$, and the match of Y, with solution $G_B[1] = g_y$, $\mathcal{M}_B[1] = Y$. Thus, B has two solutions of length 1 and 2. At gate C, all three library patterns match, generating the following solutions:

- Match of Z: This is straightforward, with the solution being $G_C[1] = g_z$, $\mathcal{M}_C[1] = Z$.

- Match of Y: In this case, the input to the match is A, and the only solution available is of length 1. Hence, the corresponding solution for C, of length 2, is $G_{\mathrm{C}}[2] = g_x \cdot g_y$,$\mathcal{M}_{\mathrm{C}}[2] = \mathrm{Y}$.
- Match of X: The input to this match is B, which has two solutions. Each of these leads to two solutions for C, of length 2: $G_{\mathrm{C}}[2] = g_y \cdot g_x$, $\mathcal{M}_{\mathrm{C}}[2] = \mathrm{X}$ and of length 3: $G_{\mathrm{C}}[3] = g_x^3$, $\mathcal{M}_{\mathrm{C}}[3] = \mathrm{X}$.

Note that we now have two solutions at circuit node C of length 2, due to the matches of Y and X. We store the one with the minimum value of cumulative logical effort.

As we have reached the primary output, the matching step is complete. We have three possible solutions at the primary output, of lengths 1, 2 and 3. The load C_L is known for each solution, and assume that the primary input has a fixed drive capability of C_{in}. This fixes the electrical effort $H = \frac{C_L}{C_{in}}$, and we can calculate the minimum delay corresponding to each available solution using Equation (5.2), and select the minimum. As discussed before, the individual gate sizes can then be determined using Equation (5.3).

Traditional approaches calculate and store solutions for all possible load values. We trade this off with generating solutions for different values of path length, N. Legal values of N depend on the library, and it is usually the case that the number of values for N is small. Thus, keeping track of N solutions is still faster than keeping track of solutions for each load value at every gate.

In order to prove the optimality of Algorithm 5.1, we state and prove the following lemma.

Lemma 1. *Let $\mathcal{I}_c$ be the critical input of a gate t, as defined previously. After sizing t and its outputs, $\mathcal{I}_c$ is still the critical input of t.*

Proof. We first prove the case of symmetric gates, in which the delay characteristics of each input pin to the output of the gate are the same. The proof for the case of asymmetric gates is similar, and follows from the proof for symmetric gates.

Consider the situation where we have a match at some gate t, with r inputs, $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_r$, each having cumulative logical effort for path of length n from the primary inputs $G_{\mathcal{I}_1}[n], G_{\mathcal{I}_2}[n], \ldots G_{\mathcal{I}_r}[n]$. Since gate t is symmetric, the load being driven by each of $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_r$ is equal, and is c_{in_t}. Let $\mathcal{I}_c$ be the critical input, and let $\mathcal{I}_j$ denote the other non-critical inputs. As mentioned previously, $\mathcal{I}_c$ being the critical input implies that $G_{\mathcal{I}_c}[n] \geq G_{\mathcal{I}_j}[n] \ \forall \ j$. In this case, we select $G_{\mathcal{I}_c}[n]$ to be multiplied with the gate effort of the match at t, g_{m_t} in order to obtain $G_t[n+1]$. This means that when the segment is sized, the size of the match at gate t (which determines the load c_{in_t} at the output of any $\mathcal{I}_j$) will be determined by the value of $G_{\mathcal{I}_c}[n]$, and this size will be different from the size determined if we had selected $G_{\mathcal{I}_j}[n]$. We show that this is in fact the correct choice to make.

As mentioned previously, the sizes of gates are determined by applying Equation (5.3) in a backward traversal. If the load at the primary output is C_L, and there are k gates from gate t to the primary output in the mapped solution, Equation (5.3) can be applied to each gate successively, to obtain

$$c_{in_t} = \frac{\prod_k g_k}{\hat{f}^k} \cdot C_L \tag{5.4}$$

Note that the stage effort for optimal delay is in the denominator of Equation (5.4), and $\hat{f} = F^{\frac{1}{N}} = (G \cdot H)^{\frac{1}{N}}$. Therefore, choosing $G_{\mathcal{I}_c}[n]$ induces a size on gate t that is smaller than that we would have obtained by using $G_{\mathcal{I}_j}[n]$. This means that the delay $G_{\mathcal{I}_j}[n]$ would have induced (say $D_{\mathcal{I}_j}$) would depend on a *larger* size of gate t. Since t is now smaller than anticipated by gate $\mathcal{I}_j$, its load on $\mathcal{I}_j$ is smaller, and hence the delay of the branch from an input to gate $\mathcal{I}_j$ does not increase (from the value it would have been, if the solution corresponding to $\mathcal{I}_j$ had been used to size t) by taking the choice of $G_{\mathcal{I}_c}[n]$, i.e., $\mathcal{I}_c$ is still the critical input of t.

We now turn to the general case of asymmetric gates. In this case, the delay of each input of the gate to the output can change depending on the functionality. However, we show that the assertion of Lemma 1 still holds.

Consider asymmetric gate t with two of its inputs, a and b having logical efforts g_a and g_b respectively. Let the cumulative logical effort up to each input be G_a and G_b, and the cumulative logical effort from the output of t to a primary output be G_k. Since gate t is asymmetric, the capacitance at each input is different, but this also implies that the logical efforts are in the same ratio i.e., if $c_{in_b} = z \times c_{in_a}$, then $g_b = z \times g_a$ (this follows from the definition of logical effort). There are two cases to consider, as follows.

1. $G_a > G_b$ In this case, the solution at input a is selected as it is considered to be critical. Sizing gate t according to this solution will imply some capacitance $c^*_{in_b}$ at input b, and we need to show that $c^*_{in_b} \leq c_{in_b}$.

$$\begin{aligned}
c_{in_a} =& \frac{G_k \cdot g_a}{\hat{f}^k_a} \cdot C_L \\
c^*_{in_b} =& z \times c_{in_a} \\
=& z \times \frac{G_k \cdot g_a}{\hat{f}^k_a} \cdot C_L \\
=& \frac{G_k \cdot g_b}{\hat{f}^k_a} \cdot C_L \\
c_{in_b} =& \frac{G_k \cdot g_b}{\hat{f}^k_b} \cdot C_L
\end{aligned}$$

Since $G_a > G_b$, $\hat{f}_a > \hat{f}_b$ and $c^*_{in_b} < c_{in_b}$.

2. $G_b > G_a$ As in the previous case, the solution at input b is selected, which implies some capacitance $c^*_{in_a}$ at input a. We need to show that $c^*_{in_a} \leq c_{in_a}$.

$$\begin{aligned}
c_{in_b} =& \frac{G_k \cdot g_b}{\hat{f}^k_b} \cdot C_L \\
c^*_{in_a} =& \frac{c_{in_b}}{z} \\
=& \frac{G_k \cdot g_b}{\hat{f}^k_b \times z} \cdot C_L \\
=& \frac{G_k \cdot g_a}{\hat{f}^k_b} \cdot C_L \\
c_{in_a} =& \frac{G_k \cdot g_a}{\hat{f}^k_a} \cdot C_L
\end{aligned}$$

Since $G_b > G_a$, $\hat{f}_b > \hat{f}_a$ and $c^*_{in_a} < c_{in_a}$.

$\square$

In Algorithm 5.1, the value of the cumulative effort for a match at a circuit node is calculated based on the previously stored optimal values at its inputs. Naturally, the value of cumulative effort at a node will be minimum only if the value at its inputs is minimum. This optimal substructure property of our formulation, along with Lemma 1, leads to an optimally mapped solution for the entire circuit.

If we map fanout-free regions only, Algorithm 5.1 provides optimal solutions. The path effort F can be used to calculate the sizes of each match selected on the critical path. Matches that are not on the critical path can be sized once the critical path has been fixed. The non-critical paths have a certain amount of slack in the delay that they have to meet. This slack, and the fact that the critical path is now presenting a load that is smaller than previously anticipated can be used in a possible optimization, to control the area of the implementation.

5.3.2 Extensions to Multiple Fanouts

In the case of multiple fanouts, we treat the circuit as a collection of fanout-free regions. In this case, the critical input of a fanout-free region is not well defined, since the path having the maximum delay through the region may not lie on the critical path of the circuit. We therefore use a modified version of Algorithm 5.1, where instead of storing only one value of $G_t[n]$, we store $G_{s_i \to t}[n]$, where $s_1, s_2, \ldots, s_k$ are the inputs of a fanout-free region ending in t (all of s_i and t are in the fanout-free region).

We now propose a solution to the load-distribution problem, by extending techniques developed in [KS04]. The circuit is initially divided into fanout-free regions, and matches for each of the fanout-free regions are generated as described above. In a primary output to primary input traversal of the circuit, we calculate a Delay-C_{in} curve, to be defined shortly, for every input of all fanout-free regions. As before, let t be the output of a fanout-free region, and let $s_1, s_2, \ldots, s_k$ be the inputs. The Delay-C_{in} curve of s_i, $D_{s_i \to PO}$, is the minimum delay of the critical path from s_i to some primary output, for different values of input capacitance. The critical path may span multiple fanout-free regions of the circuit, and $D_{s_i \to PO}$ implicitly stores the optimal values of load at each multiple fanout point, as well as the optimal distribution of this load to each fanout.

The Delay-C_{in} curve of s_i is calculated as follows. At a primary output, the Delay-C_{in} curve consists of a single delay value of zero, for the fixed load being driven. Since the circuit is traversed from the primary outputs to the primary inputs, the Delay-C_{in} curves of each fanout of t are known. Assume that t has l fanouts, $\mathrm{F}_0, \mathrm{F}_1 \ldots \mathrm{F}_l$. The load that t has to drive is the sum of the input capacitances of each of the fanouts. Since the Delay-C_{in} curves of each fanout have been calculated, for any fanout F_j, if we select a particular input capacitance, we immediately know the minimum delay of the critical path from F_j to a primary output. The minimum delay of the critical path from s_i for some value of input capacitance $c_{in_{s_i}}$ to a primary output is composed of the minimum delay of the path within the fanout-free region (i.e., the path from s_i to t) and the maximum delay from any fanout of t to a primary output. Say we have some selection of input capacitances of each fanout of t, and since matching is complete, we can select the logical effort $G_{s_i \rightarrow t}$, electrical effort $H_{s_i \rightarrow t}$ and path length $N_{s_i \rightarrow t}$ that minimize the delay of the path from s_i to t. Adding to this value the maximum delay to the primary outputs of any fanout F_j gives us the required critical path delay *for that selection of input capacitances of fanouts.* Repeating this for every combination of input capacitances of the fanouts and selecting the minimum delay thus obtained gives us $D_{s_i \rightarrow PO}$. The crucial point here is that by considering all combinations of fanout capacitances, we directly address the load-distribution problem. For one combination of fanout capacitances, some fanout F_i may be the critical one, for another combination some other fanout F_j may be critical. This information, as well as what the sizes of the other fanouts are is stored with the Delay-C_{in} curve.

Algorithm 5.2 shows how the Delay-C_{in} curve of input s_i of a fanout-free region terminating in t can be calculated. Given an electrical effort, $H = \frac{C_L}{c_{in_{s_i}}}$, the first procedure, `Calculate` $DCurve_{s_i}$ is used to calculate the best delay of a fanout-free region from all the matchings of different lengths that have been generated. `Calculate` $D_{s_i \rightarrow PO}$ of Algorithm 5.2 determines the best load, and the best distribution of this load to all fanouts, for the given input capacitance, as described above.

Consider the circuit shown in Figure 2.1, and assume that B and C drive fixed loads at primary outputs. For different input capacitances of B and C, we can calculate the minimum achievable delay, thereby obtaining the Delay-C_{in} curves at their inputs. At

Algorithm 5.2 `Calculating the Delay-`C_{in}` Curves`

```
Calculate $DCurve_{s_i}[C_L][c_{ins_i}]$
// $s_i$ is the input, $t$ is the output of the path
for all values of path length $n$ do
    $temp = n\left[G_{s_i\to t}[n] \times \frac{C_L}{c_{ins_i}}\right]^{\frac{1}{n}} + P_{s_i\to t}[n]$
    if $temp < DCurve_{s_i}[C_L][c_{ins_i}]$ then
        $DCurve_{s_i}[C_L][c_{ins_i}] = temp$
    end if
end for

Calculate $D_{s_i\to PO}[c_{ins_i}]$
// $t$ has $l$ outputs, $\mathrm{F}_0, \mathrm{F}_1 \ldots \mathrm{F}_l$
for every combination of $c_{in_j}$ of all fanouts $\mathrm{F}_j$ do
    if the selected combination is not redundant then
        $C_L = \sum_{j=1}^{l} c_{in_j}$
        Calculate $DCurve_{s_i}[C_L][c_{ins_i}]$
        $temp = DCurve[C_L][c_{ins_i}] + \max_{j=1\ldots l} D_{F_j\to PO}[c_{in_{\mathrm{F}_j}}]$
        if $temp < D_{s_i\to PO}[c_{ins_i}]$ then
            $D_{s_i\to PO}[c_{ins_i}] = temp$
        end if
    end if
end for
```

fanout-free region A, we need to consider all combinations of input capacitances of B and C. Each such combination is one possible value of load for A. For a particular load and input capacitance, we can calculate the minimum delay in A using `Calculate` *DCurve*. Combining this with the maximum of delays to primary outputs through B and C gives us a possible point on the Delay-C_{in} curve of the input of A. Note that this delay may be replaced by a smaller one, if, for example, some other value of load at the output of A induces a delay from the input to a primary output that is less than the current value.

Although it may seem that the total number of combinations of $C_{in_{\mathrm{F}_j}}$ is large (it is in fact $O(|C_{in_{\mathrm{F}_1}}| \times |C_{in_{\mathrm{F}_2}}| \times \ldots |C_{in_{\mathrm{F}_l}}|)$, where $|C_{in_{\mathrm{F}_j}}|$ is the number of possible values of input capacitance of F_j), it is shown in [KS04] that the number of combinations that actually have to be considered is much smaller ($O(|C_{in_{\mathrm{F}_1}}| + |C_{in_{\mathrm{F}_2}}| + \ldots |C_{in_{\mathrm{F}_l}}|)$), which leads to an acceptable run time of our implementation.

Algorithm 5.2 is a dynamic programming algorithm. Note that the Delay-C_{in} curves of one fanout-free region are calculated based on the curves at its outputs, and a particular critical path delay is obtained by simply taking the combination of the delay of the fanout-free region with the maximum critical path delay of the outputs. This also exhibits optimal substructure, and hence the delay curves obtained at primary input encode globally optimal solutions to the load-distribution problem.

This procedure addresses both the components of the load-distribution problem. Firstly, the *globally* optimal output and input capacitance for each fanout-free region is determined. Secondly, the best distribution of the output load into the input capacitances of the fanout-free regions being driven is determined, thus addressing the interaction between parallel regions of the circuit.

5.3.3 Summary

The complete approach for logical effort based technology mapping addressing the load-distribution problem, called MELT ($\underline{\text{T}}$echnology $\underline{\text{M}}$apping using $\underline{\text{L}}$ogical $\underline{\text{E}}$ffort: the order of letters are suggestive of the multiple input-output-input traversals of the circuit required by our approach) is presented in Algorithm 5.3. After the first three steps, which have been described previously, we have Delay-C_{in} curves at the primary inputs of the circuit. At each primary input, the load that minimizes the delay to any output is selected. The primary inputs are are processed in decreasing order of this delay. A forward traversal from the primary inputs using the selected loads fixes the input and output capacitances and the lengths of each fanout-free region. This information, in turn can be used to select the matches of the optimal solution.

In Algorithm 5.3, there are two issues that restrict the optimality of the final solution. Firstly, the processing of each input of a fanout-free region is carried out independent of other inputs of this region. The solutions generated by different inputs may contradict each other. Secondly, circuits in general have reconvergent fanouts. The interaction between multiple, overlapping reconvergent paths is difficult to analyze efficiently. For both these cases, we use the heuristic of assuming that all paths are independent, and make the best choice available. The loss of optimality is

Algorithm 5.3 `MELT: Technology Mapping using Logical Effort`

```
Divide the circuit into fanout-free regions
PI→ PO Traversal: generate matches for each fanout-free region using Algorithm 5.1, storing optimal matches for each input of the fanout-free region
PO→ PI Traversal: calculate Delay-$C_{in}$ curves for each input to the fanout-free region using Algorithm 5.2
PI→ PO Traversal: select the optimal electrical effort for each fanout-free region, and the corresponding lengths
Covering: use the assigned output and input capacitances to generate the corresponding optimal covers for each fanout-free region
```

acceptable when compared with the alternative of calculating the exact solution.

The first step in Algorithm 5.3, that of generating matches, takes time $O(|V| + |E|) \cdot |L| \cdot |N|$, where $|V|$ is the number of nodes in the graph, $|E|$ is the number of edges, $|L|$ is the size of the library, and $|N|$ is the maximum path length. $|N|$ (which is small, on average) is replaced by the number of loads considered for each match in the case of traditional approaches. Along with the fact that our library is much smaller than what would traditionally be used, the matching step of our algorithm is significantly faster. Since we store solutions for each path length, and each input to a fanout-free region, if we denote the number of inputs to a fanout-free region by $|FI|$, the storage requirement is $O(|V| \cdot |FI| \cdot N)$. Note that this is a very loose upper bound. We show in the results section that N is relatively small, and while $|FI|$ can be exponentially large in theory ($O(2^N)$, if the entire fanout free region is a tree of 2-input gates), in practice, it is much smaller. Calculating the Delay-C_{in} curves dominates the running time of our algorithm. It can be shown that the time complexity of this step is $O(|FI| \cdot |c_{in}|^2 \cdot |FO|^2)$, where $|c_{in}|$ is the number of possible values for input capacitances, and $|FO|$ is the number of fanouts at a multiple fanout point. This bound too, is very loose, and for benchmark circuits, the running time is of the same order as that of SIS.

5.4 Results

In order to validate our approach, we have implemented Algorithm 5.3 and used it to map the ISCAS combinational benchmark circuits. These results were compared with SIS [SSL+92]. The library used for SIS was generated by calibrating INV, 2-, 3- and 4-input NAND and NOR gates on a 0.1μ technology using the Berkeley Predictive Technology Model[1] [CSS+00]. Twenty sizes of each gate were generated, for a total library size of approximately 140 elements. These gates were also calibrated in order to obtain the logical effort and parasitic delays, which constitute the library used by our algorithm, with seven elements, one for each gate type. In our approach, calculating gate sizes as described in Section 5.3 can lead arbitrary values (less than the largest gate size). In order to make a fair comparison with SIS, gate sizes are normalized to the 20 sizes of each gate that are used by SIS.

The results obtained are as shown in Table 5.1. The first column lists the benchmark circuit. The next two, under the title SIS show the best delay obtained for each circuit using the command `map -n 1` in SIS, and the corresponding running time, T, in seconds. The performance of MELT for the same circuits is as shown. On average, our algorithm generates circuits that are 25.39% faster than those obtained using SIS. Interestingly, MELT also has an area improvement of 12.65%. During the covering step, the load at multiple fanout points is accurately known, and is usually higher than that estimated by SIS. Complex gates have better delay characteristics at higher loads, as compared to the equivalent using simple gates, and consequently MELT makes greater use of complex gates. Since complex gates tend to occupy less area than the equivalent circuit composed of simple gates, we observe an overall area improvement.

As mentioned before, we determine and store matches for all values of path lengths for each input of fanout-free regions of the circuit. These are then examined to obtain the minimum achievable delays for the fanout-free regions, and therefore, having long path lengths can lead to large run times. Table 5.2 presents statistics about path lengths encountered in practice. For each circuit, the column labeled #FFR lists the number of fanout-free regions, followed by the minimum, average and maximum path

[1]Available from http://www-device.eecs.berkeley.edu/~ptm.

Table 5.1: Technology Mapping: SIS Vs. MELT

Circuit	SIS		MELT		% delay
	Delay(ps)	T(s)	Delay(ps)	T(s)	improv.
C17	79.91	0.02	65.72	0.02	17.76
C432	1611.36	0.65	795.93	1.96	50.60
C499	622.59	1.16	609.17	1.20	2.16
C880	656.54	1.07	643.43	1.21	2.00
C1355	686.39	1.50	678.19	2.35	1.19
C1908	1037.01	1.74	868.42	2.31	16.26
C2670	1885.50	2.52	868.82	3.93	53.92
C3540	2148.30	3.64	1218.21	6.20	43.29
C5315	1624.71	6.01	971.36	6.40	40.21
C6288	3020.81	6.87	2893.31	8.91	4.22
C7552	2098.08	7.35	1097.69	10.15	47.68
9symml	1218.36	0.62	346.78	0.31	71.54
alu2	1994.09	1.04	1137.77	1.47	42.94
apex6	634.67	1.89	388.89	1.75	38.73
b9	440.00	0.34	227.89	0.33	48.21
cc	312.75	0.18	157.22	0.14	49.73
cm138a	195.23	0.06	120.11	0.09	38.48
cmb	228.95	0.15	216.70	0.16	5.35
count	692.37	0.36	592.17	0.21	14.47
decod	326.95	0.12	98.50	0.14	69.87
des	3659.99	13.67	833.09	52.84	77.24
example2	691.10	0.71	331.66	0.75	52.01
f51m	753.01	0.31	344.53	0.27	54.25
frg1	472.26	0.33	379.31	0.27	19.68
i5	392.54	1.12	222.76	1.03	43.25
pair	1232.27	4.36	814.31	5.71	33.92
pcler8	397.78	0.20	331.04	0.18	16.78
t	75.45	0.02	74.92	0.01	0.70
ttt2	824.10	0.68	279.97	0.40	66.03
vda	1333.71	3.48	443.30	10.94	66.76
x1	1055.87	1.09	343.81	0.47	67.44
z4ml	301.13	0.14	198.91	0.16	33.95
Average					37.20

Table 5.2: Path Length Statistics

Circuit	#FFR	Path Length			Circuit	#FFR	Path Length		
		min.	avg.	max.			min.	avg.	max.
C17	5	1	2.20	5	cm138a	12	1	4.00	8
C432	79	1	3.11	23	count	48	1	3.23	11
C499	187	1	3.89	15	cmb	17	1	3.47	26
C880	141	1	3.62	22	decod	23	1	3.96	9
C1355	291	1	3.10	15	des	931	1	5.15	19
C1908	213	1	4.25	20	example2	127	1	3.49	15
C2670	251	1	3.95	27	f51m	29	1	3.74	20
C3540	340	1	4.14	22	frg1	29	1	1.69	13
C5315	545	1	4.11	19	i5	177	1	3.25	15
C6288	1456	1	2.83	7	pair	498	1	4.15	34
C7552	797	1	4.49	23	pcler8	42	1	3.00	12
9symml	18	1	3.36	34	t	5	1	3.00	8
alu2	73	1	3.32	33	ttt2	49	1	4.80	25
apex6	302	1	2.85	21	vda	146	1	6.34	23
b9	46	1	3.20	23	x1	82	1	2.89	24
cc	31	1	2.74	13	z4ml	21	1	2.60	11

lengths. We can easily see that though some paths can be large, the average path length is in fact between 2 and 5.

Table 5.2 also provides some intuition about the performance of our algorithm. For example, C6288 has a large number of fanout-free regions with small average path lengths, as compared to other circuits. For small path lengths, the effect of varying the electrical effort is limited, which in turn restricts the freedom that our algorithm has, and results in solutions that are very similar to those of that would be obtained by traditional methods. In contrast, C7552 has fewer, but larger fanout-free regions, which results in better mapped solutions.

5.5 Conclusion and Future Directions

This chapter presents a new approach to technology mapping, based on the theory of logical effort. Most of the improvement obtained by our algorithm is due to the solution of the load-distribution problem, which allows for accurate assignment of capacitances at multiple fanout points. This leads to better selection of matches, since the exact load to be driven is known. We observe an average improvement of 32.48% in terms of delay, as compared to SIS.

In [LWGH95, LWGH97], all possible decompositions of circuits are considered during the matching step. The algorithm divides the circuit into disjoint *ugates*, and applies technology mapping to each such *ugate*. Our algorithm can be extended to generate matches in each *ugate*, and calculate Delay-C_{in} curves by traversing the ugates. This approach can also be applied to DAG-mapping [KBS98], which allows matches across tree boundaries, and therefore can generate better solutions. Here, multiple fanout points are not well defined initially. However, once the matching has been done, the fanout points are specified and the Delay-C_{in} curves can be calculated as before.

6 Libraryless Technology Mapping for Silicon-on-Insulator Technology

We present a technology mapping algorithm for implementing a random logic gate network in domino logic. The target technology of implementation is Silicon on Insulator (SOI). SOI devices exhibit an effect known as Parasitic Bipolar Effect (PBE), which can lead to incorrect logic values in the circuit. Our algorithm solves the technology mapping problem by permitting several transformations during the mapping process in order to avoid PBE, such as transistor reordering, altering the way that transistors are organized into gates, and adding pmos discharge transistors. We minimize the total cost of implementation, which includes discharge transistors required for correct functioning. Our algorithm generates solutions that reduce the number of discharge transistors required by 53%, and reduces the size of the final solution by 6.3% on average. We compare our results with a modification of a current technology mapping algorithm for bulk CMOS domino logic that reduces the cost of the final solution, and find that our algorithm outperforms this method.

6.1 Introduction

As the scaling of bulk CMOS proceeds along the roadmap, interest in Silicon on Insulator (SOI) as an alternative technology has increased. In addition, manufacturing processes have matured enough to allow large circuit implementations in SOI at acceptable defect levels. However, current algorithms used for implementing circuits in bulk CMOS are inadequate for SOI. The best approaches and traditional design techniques from bulk CMOS could be disastrous if applied to SOI. An example is the use of precharge transistors in bulk CMOS, to offset the charge sharing effect. As we will show in section 6.3.2, if precharge transistors are used in SOI, we will possibly obtain circuits that do not function correctly. Current EDA techniques too do not adequately address the needs of SOI design, and there is a requirement for new algorithms and tools targeted towards SOI designs, since simple modifications to existing algorithms by adding post-processing steps leads to solutions that are sub-optimal, as shown in later sections. This chapter address the libraryless technology mapping problem in the context of SOI. We present an algorithm that maps an arbitrary two-input logic gate network to domino logic in a manner that eliminates the "Parasitic Bipolar Effect" (PBE) [LCJ+97, Chu98] by applying transformations such as reordering transistor stacks in the gate, altering the structure of the gates to reduce their susceptibility to PBE, and inserting pmos pre-discharge transistors at appropriate points in the circuit. We take an approach of using metrics for area and delay that will generally be acceptable for any SOI implementation. The corresponding loss of detail is compensated by the reduction in complexity of the algorithm. The mapping step can be followed by a post-processing step that is specific to a given set of SOI technology parameters, possibly including transistor sizing, which our work does not address. During the mapping, the algorithm minimizes the cost of the implementation: for example, for an area objective, it would minimize the total number of transistors, including pre-discharge transistors. The techniques that we use to control PBE operate by ensuring that the body voltage of the SOI device never becomes very high, so that PBE is never triggered. This yields an added side benefit of reducing the timing hysteresis exhibited by SOI circuits due to variations in the body voltage. In narrowing the range of permissible voltages for the body (to reduce the PBE), we make the timing behavior of the circuit more predictable.

This chapter is organized as follows. We briefly introduce SOI and domino logic, and present problems typical to SOI implementations and solutions to these, with emphasis on overcoming PBE. We then present our algorithm, which performs the technology mapping specifically taking PBE into consideration. We also present a simple alteration to the technology mapping algorithm currently used for mapping into bulk CMOS domino logic that reduces the number of discharge transistors required. The efficacy of our approach, and extensions to it are shown in the results section, wherein we present cost functions that minimize the area and delay of the solution. We conclude with directions for future work.

6.2 Background

6.2.1 Silicon-on-Insulator

SOI has long been used in a variety of fields, such as radiation-hardened and high-voltage applications [BR00], [KS98]. SOI circuits have attractive properties as compared to bulk CMOS, such as reduced source- and drain-to-substrate capacitances, no body effect in series stacks of transistors and suitability for reduced V_{dd} operation for given performance [CP99, Ant97]. In addition, due to reduced capacitances, SOI devices consume less power [TCW97, JCC97]. Moreover, since transistors are isolated from each other by an insulator, they require smaller area. In spite of being smaller, faster and consuming less power than bulk CMOS, SOI has not found widespread use in the VLSI community until recently. However, recent advances in manufacturing processes coupled with a realization of the limitations of bulk CMOS technology have led to a renewed interest in SOI. Increased understanding of how SOI devices behave, and possible solutions to their quirks has lead to a wider acceptance of SOI in the VLSI community. Recently, SOI has been used in a number of high end microprocessor designs, e.g. IBM Power PC [AAC+99, BAC+00], HP-PA 8700 [HP00], and others [KPK+99, CAC+99], as well as other high performance logic circuits [STMTS00, HL01, MKA+01].

The manufacturing process of SOI is very similar to that of bulk CMOS. One of the processes used for SOI fabrication is SIMOX [KS98]. The preliminary step is to implant a layer of silicon dioxide beneath the surface of the silicon wafer. This is the "Insulator" in Silicon-on-Insulator. Transistors are created by masking and doping exposed regions on the layer of silicon above the silicon dioxide. Once transistors are fabricated in this manner, they are isolated from other devices by another layer of silicon dioxide, called Shallow Trench Isolation (STI). This structure is shown in figure 6.1. Due to this structure, the bodies of individual transistors are electrically isolated from the rest of the circuit, unlike bulk CMOS circuits where the body is

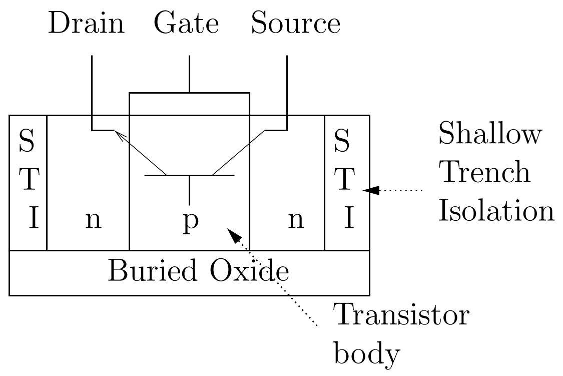


Figure 6.1: SOI Transistor Fabrication, and Presence of the Parasitic Bipolar Transistor

identical to the substrate or well, which is connected to a supply node. Hence, the body potential in SOI is free to seek its own level [WSA98], and is determined to a large extent by the voltage levels at the source and drain of the transistor, due to leakage currents. Changes in the gate voltage also affect the body potential due to capacitive coupling. Thus, if the gate is held low and drain and source are at a logic high for an extended period of time, charge accumulates in the body due to leakage current and impact ionization [Ant97]. This causes the body potential to increase. This variance in body voltage is the main source of problems associated with SOI. The change in body voltage of a device results in different switching speeds at different time instants. Also, switching speeds across a circuit can vary due to different devices having different body voltages. Another problem that a high body voltage can cause is called the Parasitic Bipolar Effect (PBE), and is described in more detail in section 6.3.

6.2.2 Domino Logic

Domino logic [KLL82] is a favored approach for implementing timing critical circuits due to their high performance. The basic structure of a domino gate is as shown in figure 6.2(a). During precharge, the dynamic node is charged to a high logic value, and the output of the gate is set to logic zero. During the evaluate phase the *n*-clock transistor switches on, and depending on the inputs to the pull down network, the dynamic node is either discharged or retains its charge. If the dynamic node switches, the output of the gate goes to a logic high value. OR logic functionality is obtained in domino by connecting n-transistors in parallel in the pull down network. Similarly, AND functionality can be obtained by connecting n-transistors in series. More complicated logic operations are obtained by combining these basic operations. An example circuit, shown in figure 6.2(a), implements the logic function $(A + B + C) * D$.

Note that the n-clock transistor shown in the figure is only required for domino gates connected to primary inputs. The outputs of domino gates change only during the evaluate phase of the clock cycle, during the precharge phase they are held low. Hence, any n-transistors driven by these outputs will be off during precharge. For a domino gate, this means that if its inputs are coming from another domino gate, the pull down network will not switch on during precharge, and hence there is no need for an n-clock transistor. Such a configuration is sometimes referred to as footless domino.

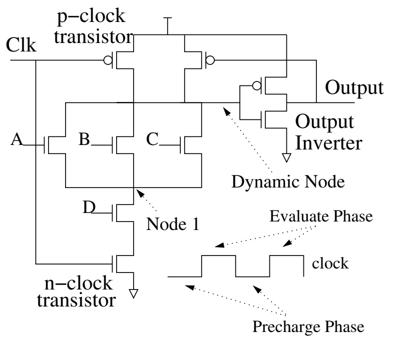
6.3 Parasitic Bipolar Effect in SOI

6.3.1 Issues with SOI Implementations

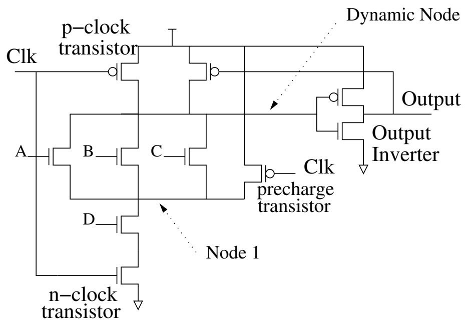
The advantages of SOI listed in the previous section come at a cost. Prominent among these are the hysteretic V_t variation [PC98], in which the behavior of a transistor varies according to its previous switching history. Another serious problem associated with SOI devices is the PBE, described in more detail in the following sections.

6.3.2 Parasitic Bipolar Effect

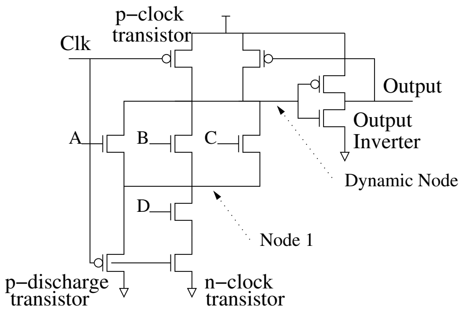
PBE occurs in certain circuit topologies and switching patterns, such as stack OR-AND structures. The topology typically involves an off transistor situated high in the stack, with the source and drain voltages in the *high* state. Over a period, this causes the body voltage to be *high*. When the source is subsequently pulled down, either by the clocked evaluation transistor in dynamic circuits or by an input signal, a large forward body bias is developed across the body-source junction, causing bipolar



(a) An Example of
a Domino Gate



(b) Precharge Transistor,
used to Avoid Charge
Sharing in Bulk CMOS



(c) Avoiding PBE
using a Predischarge
Transistor

Figure 6.2: Domino Gate Structure, and Modifications Applied in Bulk CMOS and SOI Technologies

current to flow through the lateral parasitic bipolar transistor (shown in figure 6.1). The parasitic bipolar current and the FET current (caused by noise and aggravated by the low Vt) result in a loss of charge on the dynamic node.

This can be illustrated by an example from [LCJ$^+$97]. For the circuit shown in figure 6.2(a), consider a steady state condition with inputs A = 1, B = 0, C = 0 and D = 0. Transistor A is on, and the other n-transistors in the pulldown logic network are off. Hence, as the dynamic node charges to a high value during precharge, node 1 is charged to a potential of $V_{dd} - V_{threshold}$. Recall that transistors B and C are off at this point. Under this set of conditions, the bodies of transistors A, B and C charge to a *high* value over a sufficiently large period of time. Now if signal A switches *low*, the potential at node 1 remains at its high logic value since transistor D is off. Moreover, the switching event on A sets the body voltage for device A to be low (due to strong capacitive coupling), but leaves the body voltages of B and C to be high. In the evaluate phase, if D is switched on (with A, B and C off), node 1 is suddenly pulled down. This causes the parasitic bipolar transistor to switch on, since the base and collector of the parasitic transistor are high while the emitter has been pulled low, and a large current can flow through transistors B and C. If this current is large enough, it can pull the voltage at the dynamic node to a level small enough to switch the output of the gate to a *high* value. Thus, even though the output node should have evaluated to *low*, it ends up as a *high*. In this manner, the PBE can result in a wrong evaluation if not accounted for in an SOI implementation. This value will eventually be brought to its correct value by the keeper, but this is liable to take time and may cause erroneous circuit behavior temporarily, or even permanently if any state bits are altered in the interim.

It is interesting to note that this is a typical configuration in bulk CMOS implementations that requires the use of precharge transistors, as shown in figure 6.2(c). In bulk CMOS, charge sharing is a significant problem, and precharge transistors are used to ameliorate its effects at the cost of a slight performance penalty. If any of the transistors A, B or C are on during evaluate, with transistor D off, charge on the dynamic node may be distributed to Node 1, and the potential on the dynamic node may drop low enough for the output inverter to switch erroneously. In figure 6.2(c), a precharge transistor controlled by the clock connects node 1 to V_{DD}. This transistor charges the intermediate node 1 to a high value during precharge, and redistribution of charge does not occur. SOI circuits, however, exhibit much lower drain and source capacitances, and charge sharing is not as problematic as the parasitic bipolar effect. Using precharge transistors rather than predischarge transistors as in the example above, will almost guarantee breakdown of correct circuit operation even without the sequence of transistor switchings described above.

6.3.3 Solutions to the PBE

There are a several solutions for handling the PBE, and we will enumerate these as follows:

1. The keeper pmos device can be sized up to provide some resistance to the PBE, but such a choice comes at the expense of a performance penalty due to the increased capacitance that it presents at the dynamic node and particularly at the output node.

2. Body contacts connected to the ground lines in the case of nmos, or V_dd in the case of pmos transistors can be added selectively to some devices in the circuit, but this results in an increased area and input capacitance, and is a choice that is generally avoided by SOI circuit designers [BR00].

3. If the cost is acceptable, parallel stacks can be broken up by transistor replication. For example, $(A+B+C)*D$ can be re-implemented as $A*D+B*D+C*D$ (D is replicated three times in this example). If this implementation is connected to ground, there are no paths for transistor bodies to charge high, since parallel stacks have been eliminated. A drawback of this approach is the cost requirement of duplicating logic for each finger of a potentially wide parallel stack.

4. The stack of transistors in a gate may be reordered to reduce its susceptibility to the PBE. For the gate in figure 6.2(a), if the parallel stack of transistors A, B and C is moved to the bottom of the gate, so that the sources of all three transistors are connected to ground, it will not be possible to excite the PBE. This approach exploits the reduced charge sharing effect and reduced delay dependency on stack ordering in SOI technology.

5. The above procedure works only if there is only one parallel stack per gate. If this condition is not met, it may be possible to remap the Boolean logic to the gates to ensure that each gate contains no more than one parallel stack, which can then be reordered within the gate to connect it to ground.

6. Intermediate nodes in a stack may be predischarged in every clock cycle. In figure 6.2(b), a clock-driven *p*-discharge transistor has been added to the circuit. Such a transistor can be used to connect intermediate points in the circuit (such as node 1) to ground. Thus, during every precharge cycle these intermediate nodes are discharged, and the bodies of transistors in the pulldown network are not permitted to charge to a high voltage level. The drawback of using *p*-discharge transistors is the additional load on the clock network.

7. Complex domino structures with the output inverters replaced by static NAND or NOR gates may be used to break up large parallel logic trees [CP99].

One approach to performing these optimizations is to start with the original design in bulk silicon, analyze it to identify potential sources of the PBE, and apply the above transformations to eliminate them. The main criticism of such an approach is that the solutions obtained are local in nature. For example, while a particular mapping may be optimal for bulk CMOS, it becomes non-optimal if it requires a large number of *p*-discharge transistors. A better approach would be to perform the mapping from logic gates to the transistor level, *keeping the requirement of p-discharge transistors in mind*. In section 6.5 we propose an algorithm that performs such a mapping.

In this work, we avoid the first three transformations of sizing the keeper, adding body contacts and splitting parallel stacks using duplication, since they can cause significant cost increases, and instead, focus on the rest. We perform our procedure at the time of synthesis, prior to circuit sizing, and note that the transformation that sizes the keeper is more appropriately applied after or during the transistor sizing step. In applying the remaining transformations, we will penalize the addition of clock-connected transistors and additional transistors required due to gate reorganization, since they represent a cost-increasing transformation. Reordering changes delay, but since diffusion capacitances are relatively low, we ignore them as a first order approximation.

6.4 Technology Mapping for Domino Logic

Synthesis of domino circuits is more complicated than that of static circuits. The added complexity is due to the monotonic nature of domino logic which forces it to implement only non-inverting functions. Therefore, domino logic can only be mapped to a network of non-inverting functions, where needed logic inversions must be performed at either primary inputs and/or primary outputs. Any random logic network can be transformed into a network of non-inverting functions by finding a unate network representation[1].

Generating a unate network from a binate random logic network may require logic duplication since both positive and negative signal phases may be needed. An algorithm for finding the minimum logic duplication necessary when transforming a binate random logic network into an inverter free unate network has been developed in [PBR96]. Binate-to-unate network conversion will at most double the amount of original logic (with typical overheads being much smaller) and will not increase the number of logic levels.

However, in order to avoid the complexity of [PBR96], we use a simple bubble pushing algorithm to generate the unate network. In our implementation we simply attempt to push inverters as far back as possible (i.e., towards the primary inputs), by applying DeMorgan's laws where necessary. If inverters cannot be pushed through a gate, e.g., when both positive and negative phases of a signal are required, logic duplication is necessary. After a unate network representation has been created, the network can then be technology mapped to domino gates. Note that starting from an initial decomposed network consisting of 2-input AND-OR gates and inverters, the unate network thus obtained will only consist of 2-input AND-OR gates, since all inverters have been removed in the unating process.

Technology mapping has traditionally used library based methods. In [ZS98], the authors presented a library free algorithm for technology mapping. Library free approaches have the advantage of searching a large solution space, while library based

[1]A *unate* network is one where all signals transitions occur in one direction only, either high-to-low or low-to-high. For domino logic to function correctly, all inputs to a domino gate can only make a single low-to-high transition during the evaluate cycle. Hence only unate functions can be mapped to domino logic.

methods are restricted by the size of the libraries. Since nmos pulldown networks for domino gates can be larger than their static counterparts, any precharacterized library can only explore a fraction of the exponential number of possible gate functionalities. A parameterized library overcomes this limitation although it is restricted by the use of more approximate delay models. Parameterized libraries have been used successfully to design industrial circuits, e.g., in [BF98].

The algorithm of [ZS98] uses a dynamic programming based approach. A set of tuples[2] of $\{W, H, C\}$ (width, height and cost corresponding to a pull down network configuration) are associated with each logic gate of the network. The cost here may be the number of transistors, the number of logic levels, or the delay. The values of maximum gate width and height determine the number of tuples associated with each gate. The input network of 2-input AND-OR nodes is traversed from primary inputs to primary outputs, and sub-solutions for each node for all possible configurations of $\{W, H\}$ are calculated based on the sub-solutions of its inputs. Note that, depending on the inputs, a gate may not have all combinations of $\{W, H\}$ and in practice, only a fraction of $W_{max} \times H_{max}$ tuples are associated with each gate. When calculating the sub-solution of a node, all permissible configurations of the input nodes are enumerated, and the best ones are selected. Once all valid tuples for a node have been calculated, the $\{1, 1\}$ tuple is constructed by selecting the best (lowest cost) sub-solution for that logic gate, and converting this partial structure into a domino gate by adding the clock transistors, the output inverter and a keeper transistor. Thus, the cost of a $\{1, 1\}$ configuration is the lowest cost among all other configurations plus 5. The basic operations for combining input tuples to form the tuples of the current node are AND and OR. These operations are as follows. An AND operation requires a series connection of inputs. Hence, the $\{W_1, H_1\}$ and $\{W_2, H_2\}$ solutions of the input nodes are combined to form the $\{\max(W_1, W_2), H_1 + H_2\}$ solution. Similarly, $\{W_1, H_1\}$ and $\{W_2, H_2\}$ solutions of the inputs can be combined as $\{W_1 + W_2, \max(H_1, H_2)\}$ for an OR node. A more detailed explanation of combining inputs of an AND and OR node (along with our enhancements) is presented in the next section. The algorithm is

[2]An *n-tuple* is simply a set of n ordered elements. In [ZS98], 3-tuples are used as explained in the text, in our work we associate 6-tuples with intermediate solutions as explained in the following sections.

described above illustrated briefly in Algorithm 6.1. For further details, the interested reader is referred to [ZS98].

Algorithm 6.1 `Technology Mapping for Domino Circuits`

```
// Process each node in topological order from inputs to outputs
for each node $n$ whose inputs have been processed do
  for each $\{W, H\}$ configuration of the inputs do
    if $n$ is OR then
      $\{W_{new}, H_{new}\} = \{W_1 + W_2, max(H_1, H_2)\}$
    end if
    if $n$ is AND then
      $\{W_{new}, H_{new}\} = \{max(W_1, W_2), H_1 + H_2\}$
    end if
    if $\{W_{new}, H_{new}\}$ is a valid configuration then
      $cost_{new} = cost_1 + cost_2$
    end if
    if $cost_{new}$ is better than the original cost then
      update $cost$ for configuration $\{W_{new}, H_{new}\}$
    end if
  end for
  $\{1, 1\}$ = convert configuration with lowest cost into a gate
end for
```

This algorithm guarantees optimal-cost solutions. Note that the best sub-solution of a input node may not necessarily end up as part of the final solution. Thus, local optimal solutions are avoided if they are not globally optimal. Finally, at the primary outputs, the best solution in terms of the cost function is selected. This specifies a domino circuit that implements the input network logic with minimum cost. The cost function in the above algorithm has been taken to be the total number of transistors in the implementation, but this may also be modified to minimize the maximum number of levels from primary inputs to primary outputs, in order to reduce the maximum input-to-output delay of a domino implementation.

This algorithm is easily illustrated with the help of an example. Consider the circuit in figure 6.3, and assume that the maximum number of transistors allowed in series and in parallel are 4. This simple circuit consists of 2 AND nodes and 1 OR node. The AND nodes are driven by the primary inputs, which have only

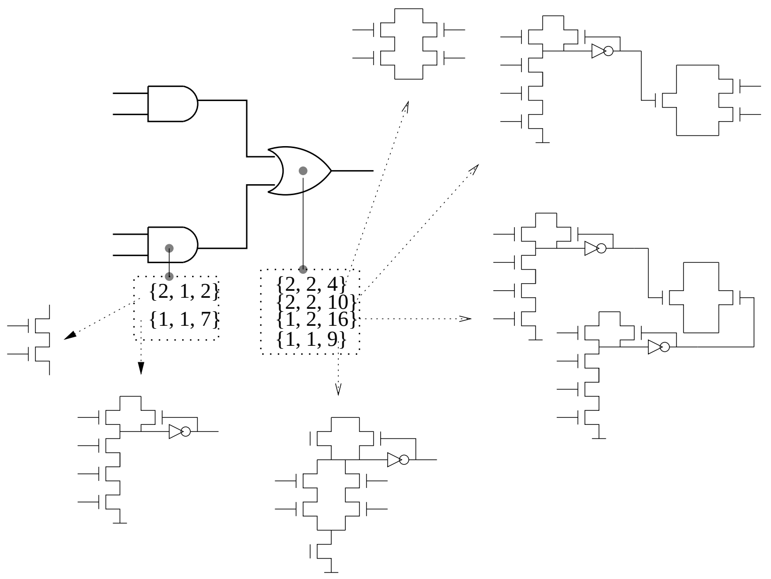


Figure 6.3: Technology Mapping for Domino Logic

one possible tuple associated with them : {1, 1, 1}. These can be combined in an AND operation to form the tuple {2, 1, 2}, for which the transistor structure is as shown. Since there is only one tuple for this gate, it is used to construct the tuple corresponding to $W = 1, H = 1$, $\{1, 1, 7\}$. The two solutions for each of the AND nodes can be combined in 4 possible ways, but due to symmetry we have only three unique combinations - {1, 2, 16}, {2, 1, 10} (repeated twice) and {2, 2, 4}. Note that when a *gate* from an input node is used (corresponding to the {1, 1} solution), an extra transistor is needed in the next level. For the OR node, the {2, 2} solution is clearly the best, and it is used to form the corresponding {1, 1} solution, with a cost of 9.

We use this basic approach in our algorithm with modifications to the cost function calculation in order to properly account for the PBE.

6.5 An Algorithm for SOI Mapping

We follow the basic algorithmic framework of [ZS98], presented in brief in section 6.4. As before, each node in the input network is associated with a set of tuples corresponding to one $\{W, H\}$ solution of the subtree rooted at the current node. W and H represent the width and height of the pull down network of the domino gate; the maximum values are user-specified.

Our objective in this work is to reduce the number of discharge transistors required to avoid PBE. An area objective (in terms of number of transistors) follows logically from this as the cost function to minimize. Hence we choose our initial cost associated with each tuple to be the number of transistors required to implement the logic correctly, as well as to avoid PBE. A delay objective can also be used as the cost function; in this case the actual cost function is a combination of delay and number of discharge transistors used.

In addition to the cost associated with each tuple, we also store p_{dis}, the number of *potential discharge transistors* required by the configuration, and par_b, which tracks whether or not a given tuple has a parallel branch at the bottom of its structure. The potential discharge transistor count is used to guide tuple combination and gate formation. Depending on the actual operation performed (i.e., AND or OR), p_{dis} is converted to actual discharge transistors, else its value is propagated to the next level. Since OR is the only operation that introduces parallel stacks, par_b is set to true in an OR operation and is propagated in an AND operation depending on the combination of input tuples (this will be explained in greater detail shortly). As mentioned in the previous section, the solutions of the input gates are combined to form the solutions for the current node. In case of multiple solutions being available, the lowest cost solution is selected. Ties for the lowest cost solutions are resolved by the p_{dis} values.

We now explain the concepts of p_{dis} and par_b, which are central to our algorithm. The parameter p_{dis} is used to account for the discharge transistors that will have to be added to eliminate the PBE. From the explanation of section 6.3.2, we see that the PBE can be excited only in the presence of one or more parallel stacks. This provides a path for the bottom of the stack to get charged to a high value (the top of the stack is charged via a path from the precharge transistor). Additionally, at least one transistor is required beneath a parallel stack to excite the PBE; when this transistor switches on, the common node for the stack will be pulled low, possibly resulting in the PBE. Hence, the bottom of a parallel stack is one potential discharge point. The parameter par_b keeps track of whether a given intermediate structure

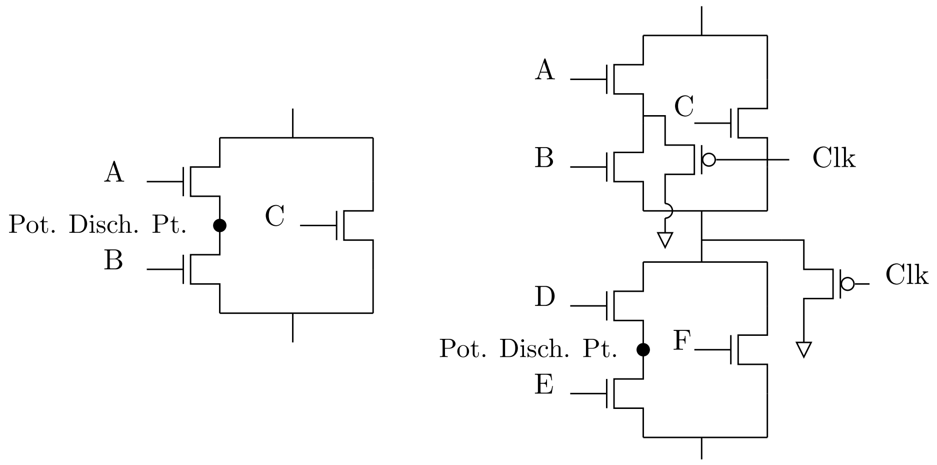


Figure 6.4: Potential Discharge Points and p-discharge Transistors

has a parallel branch at the bottom or not. In the final solution, if this point is connected to ground, no discharge transistors are required. On the other hand if it is *not* connected to ground, all intermediate points as specified by p_{dis} will have to be discharged. Hence, in an OR operation, we set par_b to true to account for the presence of a parallel stack. For an AND operation, it is set to the value of the tuple being placed at the bottom of the stack. In addition, we conditionally increment p_{dis} by one for an AND operation, since the intermediate point in a series stack may have to be discharged. In figure 6.4(a), the series connection of $A * B$ has introduced an intermediate discharge point. If $A*B$ were converted to a domino gate, or combined with other transistors in series, there would be no need to discharge this point. However, if it is connected in parallel with another configuration (as shown in the figure), this point becomes a potential discharge point for the OR tuple too - which will have to be discharged if the OR configuration is not connected directly to ground. Intermediate points in OR structures have to be discharged because of the following possible scenario. When A = 0, B = C = 1, there is a path form the top of the stack to the drain of transistor A. The source and drain of transistor A can now potentially go *high*, causing the body voltage of A to increase and thus leading to PBE.

Now consider a more complex case. Let us assume that two structures of the form shown in figure 6.4(a) have to be ANDed together - $A * B + C$ and $D * E + F$.

Each of them has 1 potential discharge point, at the junction of A and B, and D and E. The AND operation will introduce one more potential discharge point. However, when these two parallel stacks are connected in series, the structure on the top will never be connected to ground. Hence, its potential discharge points always have to be discharged by the addition of p-discharge transistors. In addition, the intermediate point introduced by the AND operation also has to be discharged. This is shown in figure 6.4(b). To sum up, for an AND operation we need to perform the following computation -

$$\begin{aligned}
p_{dis} &= p_{dis^{bottom}}; \qquad (6.1)\\
\text{discharge transistors} &= p_{dis^{top}} + 1;\\
cost &= cost_{bottom} + cost_{top} + \text{discharge transistors};\\
par_b &= par_{b^{bottom}};
\end{aligned}$$

Note here that the *cost* of a particular tuple includes not only the cost of implementing the logic, but also the discharge transistors required for avoiding PBE. Thus, when we select a lowest cost solution from various available solutions, we obtain an implementation that minimizes the cost while simultaneously avoiding PBE. The cost may be area, measured in terms of the total number of transistors required to correctly implement the required functionality, or the delay, in terms of the number of levels traversed by an input signal.

This leads to another interesting optimization that is used in our algorithm. Since our aim is to minimize the cost of the implementation as well as the total number of discharge transistors used, we can use the information implicit in p_{dis} and par_b to determine which input tuple is on the top in the series connection and which is on the bottom. If only one input has a parallel branch, we place this at the bottom, in the assumption that it could potentially be connected to ground. However, if both inputs have $par_b == true$, i.e., both inputs have parallel branches, the tuple order is determined by p_{dis}. We select the tuple with the larger p_{dis} to be at the bottom of the stack since this introduces fewer discharge transistors (ofcourse, all of these calculations are made under the optimistic assumption that the bottom of this

Algorithm 6.2 `Algorithm for Mapping SOI Circuits`

```
for each node $n$ whose inputs have been processed do
  if $n$ is OR then
    combine_or(inputs) ;
  end if
  if $n$ is AND then
    combine_and(inputs) ;
  end if
  if multiple tuples obtained for the same $W, H$ then
    Select tuple with lowest cost
    if costs are equal then
      Select tuple with lowest $p_{dis}$
    end if
  end if
  create_domino_gate
end for

combine_or
$W = W_{input_1} + W_{input_2}$;    $H = \max(H_{input_1}, H_{input_2})$;
$cost = cost_{input_1} + cost_{input_2}$;
$p_{dis} = p_{dis^{input_1}} + p_{dis^{input_2}}$;    $par_b = true$;

combine_and
if $par_{b^{input_1}}$ && $par_{b^{input_2}}$ then
  $top = \min(p_{dis^{input_1}}, p_{dis^{input_2}})$;
  $bottom = \max(p_{dis^{input_1}}, p_{dis^{input_2}})$;
else
  top = input with $(par_b == false)$ ;
end if
$W = \max(W_{top}, W_{bottom})$;    $H = H_{top} + H_{bottom}$;
$total\ dis\ trans. = p_{dis^{top}} + 1$;
$cost = cost_{top} + cost_{bottom} + total\ dis\ trans$ ;
$p_{dis} = p_{dis^{bottom}}$;    $par_b = par_{b^{bottom}}$;

create_domino_gate
Select tuple with lowest cost
Add $p$-clock transistor, output inverter and feedback transistor
if tuple has primary inputs then
  Add $n$-clock transistor
end if
```

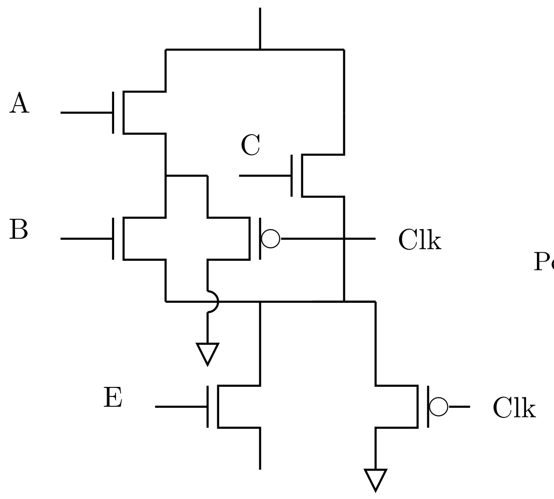


Figure 6.5: Switching Transistor Stacks, with Potential Discharge Points Highlighted

stack could potentially be connected directly to ground. If this does not happen, the ordering of parallel stacks in series is irrelevant). Consider the circuit shown in figure 6.5, wherein $A * B + C$ is to be ANDed with E. If the structure on the left is used (with E at the bottom), we have to add two discharge transistors. However, if the circuit on the right is used (with E on the top, and the parallel stack on the bottom), we have two potential discharge points, but no immediate discharge transistors. If this structure is then connected to ground, the potential discharge points will not have to be discharged, as explained previously. Note that this switching of stacks can also be done for circuits mapped for regular bulk CMOS. As we will show in the results section, using this technique as a stand-alone optimization is not as effective as our algorithm.

For an OR operation, we only need to add the p_{dis} values of the input tuples, and set the par_b parameter to $true$:

$$\begin{aligned} p_{dis} &= p_{dis^{input_1}} + p_{dis^{input_2}}; \\ cost &= cost_{input_1} + cost_{input_2}; \\ par_b &= true; \end{aligned} \tag{6.2}$$

Note that though the p_{dis} seems to function in an identical manner, for OR and AND structures, their interpretation is quite different. In both cases, p_{dis} refers to the number of points that must potentially be discharged.

However, in case of an AND, these points will have to be discharged only in case of an OR operation, for OR they will have to be discharged only if the stack is not directly connected to ground.

The algorithm is presented in listing 6.2. Each node is processed in topological order, from primary inputs to primary outputs. This ensures that the inputs of the current node being processed have been processed previously, and the corresponding sub-solutions for the inputs are available. We then combine the inputs of the node being processed in functions `combine_or` or `combine_and`, depending on the functionality of the node. These functions carry out the calculations presented previously. In addition, `combine_and` also determines the order of its inputs in the series stack as a function of the input values of par_b and p_{dis}. For multiple solutions for a given $\{W, H\}$ pair, we select the tuple with the lowest cost (which includes the number of discharge transistors). Ties on cost are split according to the value of p_{dis}.

A final comment on the algorithm is that we need to maintain two costs for each tuple. The first specifies the optimal cost if the partial structure is connected to ground, and the second if it is not. At the time of gate formation, the appropriate value is used in determining the optimal cost. For convenience, these details are omitted in the pseudocode in listing 6.2.

This algorithm is an example of a dynamic programming approach to solving an optimization problem. Each node stores all possible solutions, with associated costs, as defined by the cost function. Locally optimal solutions need not be part of a globally optimal solution, however by enumerating all possible solutions at each node we are guaranteed an optimal solution at the output. This assertion holds for all cost function that are monotonic increasing as we proceed from inputs to outputs. The cost functions that we address in this work are area and delay metrics, which also include the number of discharge transistors have this property, and hence the final solution obtained is optimal.

6.6 Results

The algorithm presented in section 6.5 has been implemented in C++ and has been tested on ISCAS benchmark circuits. In all cases, we chose the maximum width and height of the pull down network of a domino gate to be 5 and 8 respectively. Such a large value for a pull down network is valid for SOI due to the reduced source and drain capacitances. Since SOI has lower source/drain capacitances than bulk CMOS, charge sharing is not a major problem and can be handled by the weak pullup. By adding at most one p-discharge transistor at each node, we minimize its detrimental effect on charge sharing. While circuit performance does degrade slightly when compared with having a precharge transistor, this is a minor cost to pay to avoid circuit malfunction[3]. For comparison purposes, we have implemented a bulk CMOS mapping algorithm that maps circuits without regard to potential discharge points. This algorithm is referred to as `Domino_Map`. p-discharge transistors are added in a post-processing step. We compare this solution with different approaches to mapping circuits for SOI, including our algorithm with area and delay cost functions.

6.6.1 `Rearrange_Stacks_Map`

The first three columns next to each circuit name in table 6.1 show the cost associated with the solution obtained from `Domino_Map`, specifically listing the total number of domino transistors (T_{logic}), the number of pmos discharge transistors added (T_{disch}) and the sum of these two, which is the total number of transistors(T_{total}). We then ran our algorithm without regard to potential discharge points as in `Domino_Map`, but added a post-processing step that rearranges series stacks (generated by AND operations) so as to move parallel sections with a large number of potential discharge points closer to ground. The reasons for doing this have been discussed in the previous section. The solution obtained is listed in the columns under `RS_Map`. We found an average reduction of 25.4% in the number p-discharge transistors, and a 3.44% reduction in the total number of transistors. As can be seen, simply re-ordering

[3]Another option to avoiding the detrimental affect of the p-discharge transistors is to remap the logic, avoiding PBE-inducing structures. However, this will result in a larger number of domino gate levels, leading to an even larger delay.

transistor stacks leads to some decrease in the number of discharge transistors.

6.6.2 `SOI_Domino_Map`

The results of applying algorithm `SOI_Domino_Map` of listing 6.2 to the benchmark circuits are presented in table 6.2. Comparing the results obtained from `Domino_Map` and `SOI_Domino_Map`, it is clear that though the number of domino logic transistors required in SOI may be more, this increase is more than compensated by the fewer number of p-discharge transistors required, thus saving on the total number of transistors used. The average reduction in the number of discharge transistors is 53%. The last two columns list the reduction in the *total* number of transistors required for the implementation. We obtain an average reduction of 6.29%, even though the number of logic transistors (without p-discharge) has increased.

Thus, while a simple reordering of series stacks does result in some cost benefit, it is still only half the reduction of our algorithm.

6.6.3 Penalizing Clock Connected Transistors

Realizing the effects of loading on the clock network, we then applied algorithm `SOI_Domino_Map` to the same circuits, assigning a cost for the clock-driven transistors that is k times the cost of a regular transistor, where k is a user specified value. The clock connected transistors include p-clock and n-clock transistors in the domino gates along with the p-discharge transistors. On the one hand, the effect of including the cost of the p-clock and n-clock transistors of the gate is to make gate formation operation more expensive (the cost of the $\{1, 1\}$ solution for each tuple makes it less likely to be selected), and the algorithm prefers to include as many transistors in each pull down network as possible. Incrementing the cost of the p-discharge transistors, on the other hand, pushes the algorithm towards forming domino gates early, so as to avoid the overhead of the p-discharge transistors. As the results show, our algorithm chooses a result balanced between these extremes, and as the cost of clock driven transistors is increased, the solutions reduce the number of gates *and* p-discharge transistors, along with an increase in the total number of transistors required for the implementation[4]. The columns labeled T_{clock} is the number of transistors connected to the clock network. This figure is obtained by adding the number of p-discharge transistors to the clock transistors in the domino gates. The last column shows the percentage reduction in the number of clock-driven transistors, on average we reduce this figure by 3.82%. An interesting observation is that the number of clock connected transistors does not change significantly as k is varied. Also note that not much improvement is obtained for circuits that have a relatively small number of clock connected transistors. This is to be expected, since there is less freedom to choose between different solutions. Larger circuits, on the other hand, provide the algorithm with a greater number of options, and we obtain more improvement for these.

6.6.4 Depth Optimization

We now address the minimization of the *delay* of an implementation. As an approximation of the delay, we set the cost function to be the depth, i.e., the maximum number of levels of domino gates that a signal passes through from primary inputs to primary outputs. A more accurate delay model would require characterization of every possible pulldown network, and this is not feasible for a libraryless approach. As before, `Domino_Map`, reduces the number of levels required for an implementation, and discharge transistors are added as a post-processing step. In `SOI_Domino_Map`, the number of discharge transistors needed is included as a part of the cost. The results of running each of these algorithms are as presented in table 6.4. The second column next to each circuit name shows the maximum number of 2-input AND/OR gates in the original network that a signal passes through, from primary inputs to primary outputs. The columns under `Domino_Map` list the number of transistors required for implementing the logic functionality, the number of discharge transistors added in the post-processing phase, the total cost of the solution, and the number of levels in the solution. The corresponding columns under `SOI_Domino_Map` are similarly calculated using the new cost function, so that the discharge transistors are included during the mapping phase. Reducing the number of levels in an implementation drives a

[4]The figures in table 6.3 represent the number of transistors, not their weighted cost

solution towards favoring complex gates. As in the previous section, trying to reduce the number of discharge transistors drives the solution in the opposite direction, i.e., towards smaller gates. As can be seen from the results, algorithm `SOI_Domino_Map` reduces the number of levels for a few circuits, and increases them for others, in comparison with `Domino_Map`. The key result, though, is that the *sum* of number of levels and discharge transistors is reduced. We obtain an average reduction of 49.76% in discharge transistors required, and a 6.36% reduction in the number of levels.

6.7 Conclusion

We have presented an algorithm that maps gates in a logic network to a domino implementation suitable for use in SOI circuits. As the results in section 6.6 show, the lowest cost solution for domino mapping in bulk silicon technology is not an effective solution in the context of SOI. Our algorithm minimizes a specified cost function, which includes the discharge transistors required. This cost function can be as area cost or a delay cost. We also show how we can apply the algorithm by skewing the cost of clock transistors in order to reduce the load on the clock network. A similar approach can be used to derive a solution with as few gates as possible, by increasing the relative cost of gate formation. In fact, this technique can be applied to mapping for bulk CMOS too, in order to reduce the load on the clock network.

A further improvement is from the observation that the n-clock transistors are required only for gates that have primary inputs. Gates whose inputs come from other domino gates do not need these transistors, since these inputs are always low during precharge, and footless domino may be used instead. Hence, the pull down network will never be switched on during precharge. The big advantage of using this scheme for SOI is that even if a parallel stack has potential discharge points, if these are connected to ground it is not necessary to actually discharge them.

Our mapping algorithm assumes the worst case scenario, in which particular structures susceptible to breakdown have to be discharged correctly. However, breakdown will only occur for a particular sequence of input logic values. We have not taken this into account in our algorithm, and incorporating this information could lead to better solutions. In fact, this could be used for solving the hysteresis effect [PC98] in

SOI too. Once a transistor netlist has been obtained from the logic level description of the circuit as presented in this chapter, a followup technology-specific optimization step can be used to obtain further delay improvements.

Table 6.1: Comparison of Domino_Map and Rearrange_Stacks_Map

Circuit	`Domino_Map`			`RS_Map`			Reduction in T_{disch}		Total Reduction	
	T_{logic}	T_{disch}	T_{total}	T_{logic}	T_{disch}	T_{total}	ΔT_{disch}	%	ΔT_{total}	%
cm150	73	19	92	73	15	88	4	21.05	4	4.35
mux	73	21	94	73	18	91	3	14.29	3	3.19
z4ml	127	16	143	127	12	139	4	25.00	4	2.80
cordic	199	38	237	202	23	225	15	39.47	12	5.06
frg1	244	78	322	239	43	282	35	44.87	40	12.42
b9	365	87	452	367	57	424	30	34.48	28	6.19
apex7	663	124	787	662	106	768	18	14.52	19	2.41
c432	655	167	822	675	128	803	39	23.35	19	2.31
c880	1163	198	1361	1182	153	1335	45	22.73	26	1.91
t481	1448	232	1680	1458	193	1651	39	16.81	29	1.73
c1355	1856	130	1986	1856	86	1942	44	33.85	44	2.22
apex6	1889	319	2208	1896	275	2171	44	13.79	37	1.68
c1908	1924	208	2132	1924	171	2095	37	17.79	37	1.74
k2	2425	345	2770	2441	278	2719	67	19.42	51	1.84
c2670	2467	422	2889	2481	341	2822	81	19.19	67	2.32
c5315	5498	830	6328	5510	603	6113	227	27.35	215	3.40
c7552	8088	1082	9170	8138	760	8898	322	29.76	272	2.97
des	9069	1416	10485	9097	929	10026	487	34.39	459	4.38
Average								25.41		3.44

Table 6.2: Comparison of Domino_Map and SOI_Domino_Map

Circuit	`Domino_Map`			`SOI_Domino_Map`			Reduction in T_{disch}		Total Reduction	
	T_{logic}	T_{disch}	T_{total}	T_{logic}	T_{disch}	T_{total}	ΔT_{disch}	%	ΔT_{total}	%
cm150	73	19	92	73	15	88	4	21.05	4	4.35
mux	73	21	94	73	15	88	6	28.57	6	6.38
z4ml	127	16	143	127	12	139	4	25.00	4	2.80
cordic	199	38	237	206	18	224	20	52.63	13	5.49
frg1	244	78	322	245	20	265	58	74.36	57	17.70
f51m	297	71	368	309	31	340	40	56.34	28	7.61
count	333	71	404	365	22	387	49	69.01	17	4.21
b9	365	87	452	367	29	396	58	66.67	56	12.39
9symml	424	107	531	440	39	479	68	63.55	52	9.79
apex7	663	124	787	667	59	726	65	52.42	61	7.75
c432	655	167	822	706	99	805	68	40.72	17	2.07
c880	1163	198	1361	1223	81	1304	117	59.09	57	4.19
t481	1448	232	1680	1495	54	1549	178	76.72	131	7.80
c1355	1856	130	1986	1856	46	1902	84	64.62	84	4.23
apex6	1889	319	2208	1928	183	2111	136	42.63	97	4.39
c1908	1924	208	2132	1949	109	2058	99	47.60	74	3.47
k2	2446	348	2794	2527	114	2641	234	67.24	153	5.48
c2670	2467	422	2889	2498	244	2742	178	42.18	147	5.09
c5315	5498	830	6328	5510	474	5984	356	42.89	344	5.44
c7552	8088	1082	9170	8164	637	8801	445	41.13	369	4.02
des	9069	1416	10485	9122	581	9703	835	58.97	782	7.46
Average								53.00		6.29

Table 6.3: Comparison of the Number of Transistors Under Different Weights of p_{dis}

Circuit	$k = 1$					$k = 5$					%Improv
	T_{logic}	T_{disch}	T_{total}	$\#G$	T_{clock}	T_{logic}	T_{disch}	T_{total}	$\#G$	T_{clock}	
cm150	73	15	88	3	21	73	15	88	3	21	0.00
mux	73	15	88	3	21	73	15	88	3	21	0.00
z4ml	134	13	147	9	39	134	13	147	9	39	0.00
cordic	222	19	241	14	52	217	19	236	13	51	1.92
frg1	283	20	303	19	58	277	21	298	18	57	1.72
count	374	22	396	28	77	374	22	396	28	77	0.00
b9	367	29	396	29	87	373	26	399	30	86	0.11
c8	331	42	373	26	94	325	42	367	25	92	2.12
f51m	405	42	4 47	27	104	391	38	429	26	98	5.76
9symml	571	57	628	34	132	482	36	518	33	106	19.69
apex7	739	67	806	54	175	733	67	800	53	173	1.14
x1	825	63	888	65	193	816	60	876	64	188	2.59
c432	799	93	892	52	197	804	89	893	53	194	1.52
i6	1155	67	1222	67	201	1155	67	1222	67	201	0.00
c1908	992	117	1109	77	259	957	111	1068	78	254	1.93
t481	1916	77	1993	132	325	1927	70	1997	135	316	2.77
c499	2016	46	2062	130	440	2016	46	2062	130	440	0.00
c1355	2016	46	2062	130	440	2016	46	2062	130	440	0.00
dalu	2073	182	2255	158	446	2065	177	2242	158	441	1.12
k2	3127	109	3236	195	481	3142	107	3249	195	475	1.24
apex6	2418	206	2624	158	520	2516	185	2701	160	504	3.07
rot	2520	290	2810	174	627	2449	262	2711	172	595	5.10
c2670	2608	247	2855	162	642	2614	244	2858	163	641	0.15
C5315	5755	535	6290	433	1501	5754	515	6269	439	1491	0.66
c3540	6659	634	7293	427	1501	6377	552	6929	412	1393	7.93
des	9818	600	10418	594	1581	9390	493	9883	586	1453	8.09
c7552	7519	584	8103	582	1853	7376	508	7884	580	1759	5.07
Average											3.82

Table 6.4: Depth and Discharge Transistor Optimization

Circuit	L	`Domino_Map`				`SOI_Domino_Map`				Reduction in T_{disch}		Reduction in L	
		T_{logic}	T_{disch}	T_{total}	L	T_{logic}	T_{disch}	T_{total}	L	ΔT_{disch}	%	ΔL	%
z4ml	16	182	22	204	7	176	12	188	6	10	45.45	1	14.29
cm150	10	268	35	303	9	193	20	213	7	15	42.86	2	22.22
mux	10	268	35	303	9	193	19	212	7	16	45.71	2	22.22
cordic	12	373	40	413	9	310	19	329	8	21	52.50	1	11.11
f51m	30	534	75	609	25	598	49	647	20	26	34.67	5	20.00
c8	11	591	80	671	6	564	44	608	6	36	45.00	0	0.00
frg1	14	607	102	709	12	503	52	555	11	50	49.02	1	8.33
b9	10	659	106	765	9	537	47	584	6	59	55.66	3	33.33
count	21	741	76	817	7	672	56	728	9	20	26.32	-2	-28.57
c432	34	981	125	1106	26	1229	107	1336	25	18	14.40	1	3.85
apex7	17	974	139	1113	11	1111	82	1193	7	57	41.01	4	36.36
9symml	21	1038	174	1212	14	800	70	870	12	104	59.77	2	14.29
c1908	32	1292	251	1543	16	1625	167	1792	14	84	33.47	2	12.50
x1	12	1490	233	1723	9	1364	106	1470	8	127	54.51	1	11.11
i6	6	2109	237	2346	4	2143	133	2276	4	104	43.88	0	0.00
c1355	20	2640	244	2884	7	2456	44	2500	7	200	81.97	0	0.00
t481	23	2794	196	2990	17	3301	97	3398	16	99	50.51	1	5.88
rot	27	2768	514	3282	11	3259	320	3579	14	194	37.74	-3	-27.27
apex6	21	3816	584	4400	15	4222	315	4537	12	269	46.06	3	20.00
k2	21	4181	324	4505	13	3847	143	3990	12	181	55.86	1	7.69
c2670	31	4052	521	4573	16	4207	281	4488	14	240	46.07	2	12.50
dalu	23	3795	786	4581	10	2747	249	2996	12	537	68.32	-2	-20.00
c3540	42	7675	1341	9016	19	9021	601	9622	20	740	55.18	-1	-5.26
c5315	36	8216	1074	9290	17	9409	493	9902	17	581	54.10	0	0.00
c7552	42	10374	1172	11546	29	10747	501	11248	22	671	57.25	7	24.14
des	26	14068	2653	16721	14	21313	944	22257	14	1709	64.42	0	0.00
Average											49.76		6.36

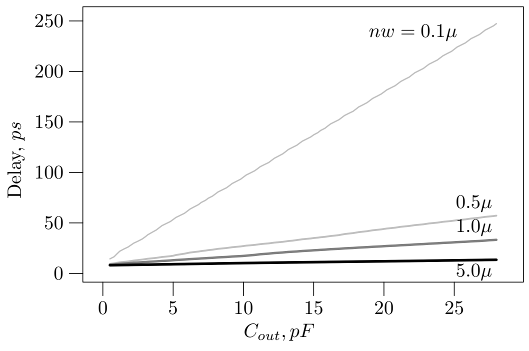
Appendix A

Generating Libraries for Technology Mapping

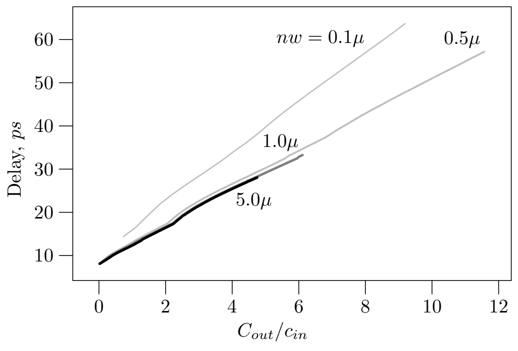
Technology mapping maps a (technology independent) netlist of logic gates to a library of cells that have been characterized in terms of parameters such as delay, area and power. In Chapter 5 we present a new approach to technology mapping, that optimizes the delay of a given circuit. In order to validate our approach, we need a library that has been characterized to obtain the logical effort and parasitic delay of each cell. Additionally, in order to verify the improvements obtainable by our approach, we need a library that can be used with the traditional approach. We target a 0.1μ technology, and cell libraries for this generation are not currently available. Thus, we need two libraries, one for the traditional approach and one that can be used by MELT (called the MELTing Library). In this appendix, we describe the approach taken in order to generate these libraries.

We describe the methodology used for calibrating inverters in detail. The other gates used in our library (NAND2, NAND3, NAND4, NOR2, NOR3, NOR4, AOI21, AOI22, AOI211, AOI221, AOI222, AOI31, AOI32, AOI33, OAI21, OAI22, OAI211, OAI221, OAI222, OAI31, OAI32, OAI33 – 22 logic functions in all) are calibrated in a similar fashion. The actual values extracted for the MELT library are then presented; the equivalent traditional technology library has 440 gates, and is not presented here.

A.1 Caliberating the Inverter



(a) Delay Vs. Load, C_{out}



(b) Delay Vs. Elec. Effort, $h = \frac{C_{out}}{c_{in}}$

Figure A.1: Inverter Delay

Figure A.1(a) shows the delays of inverters of different sizes as a function of output capacitance. This data was obtained using SPICE. The set of points labeled 0.1μ is the delay of an inverter with n-transistor width of 0.1 microns, and a p-transistor sized in order to have equal rise delay. As expected, the delay of the inverter rises as the load being driven increases. The other data sets plot the delay of inverters having different widths of the n-transistors – 0.5, 1.0 and 5.0μ, each of which have varying dependence of delay on load. A casual inspection is enough to convey the strong linearity of this dependence. A least-squares algorithm can be used to determine a linear approximation for the delay function. Rather than selecting a large number of inverters, we choose a varying granularity of inverter sizes; more small inverters and fewer inverters of large sizes. We use `gnuplot` in order to perform this approximation, and obtain the characterization shown in Table A.1. The function used is

$$delay = slope \times load + intercept \tag{A.1}$$

These values are used in the library used for traditional technology mapping. As can be seen, the error is very small, and is less than 1% for all parameters. Note

n-width	slope	% error	intercept	% error
0.1	8.43	0.05	10.97	0.64
0.2	4.21	0.09	10.31	0.62
0.3	2.83	0.15	9.86	0.69
0.4	2.13	0.21	9.59	0.76
0.5	1.72	0.26	9.34	0.79
0.6	1.45	0.32	9.17	0.83
0.7	1.24	0.40	9.07	0.89
0.8	1.09	0.49	9.02	0.97
0.9	0.98	0.52	8.86	0.95
1.0	0.89	0.53	8.70	0.90
1.2	0.78	0.47	8.41	0.70
1.5	0.64	0.40	8.14	0.52
1.8	0.54	0.51	8.11	0.55
2.0	0.48	0.55	8.16	0.52
2.5	0.36	0.36	8.35	0.26
3.0	0.30	0.38	8.33	0.23
3.5	0.26	0.40	8.27	0.21
4.0	0.23	0.43	8.22	0.20
4.5	0.21	0.46	8.19	0.20
5.0	0.19	0.52	8.17	0.20

Table A.1: Linear approximation of delay as a function of load

the small values of the slopes for larger gate sizes. This is indicative of the delay independence of load for these larger gates.

Next, consider the plots of Figure A.1(b). This is the same data as that used in Figure A.1(a), but is plotted with the electrical effort on the x-axis. An interesting trend is immediately apparent, and this is indeed the basis of logical effort – the delay of an inverter is a linear function of the electrical effort, independent of its size. The exception is the smallest inverter, and we believe this discrepancy is due to deep sub-micron effects that arise at small geometries. We can once again perform a least-squares approximation to in order to express delay as such a linear function:

$$\begin{aligned} delay &= g \times h + p && \text{(A.2)} \\ &= 4.21 \times h + 8.67 && \text{(A.3)} \\ &= 4.21(1 \times h + 2.11) && \text{(A.4)} \end{aligned}$$

We assume the logical effort of an inverter to be unity, which allows us to extract a *delay unit* $\tau = 4.21$. τ ties the theory of logical effort to a particular fabrication process. The logical effort equivalent of Table A.1 is shown in Table A.2.

n-width	g	p
any	1	2.11

Table A.2: LE-equivalent of Table A.1

Thus, we have reduced a library of 20 inverters to a library with a single inverter, which can be used in MELT. Note that the inverter could have been calibrated using more than twenty sizes; while this would increase the size of the traditional library, we would only need one gate for the library used in MELT.

A.2 The MELT Library

The logical efforts and parasitic delays of gates used in the MELTing Library are presented in Table A.3. We calibrate each input of asymmetric gates, this data is also shown. We note here that the actual values of parasitic delays are much higher than those predicted by theory. This is due to technology being targetted – for small devices, internal capacitances have a larger effect on delay than before. It is also interesting to note the clear trend of complex gates having higher values of logical effort. This is to be expected, and in general, for driving large loads on critical paths, simple gates are preferred.

A more complete library can have multiple, possibly orthogonal, dimensions, e.g., for gates with different threshold voltages, different sizes of internal transistors, etc. This increases the library size, and such gates may not be amenable to logical effort.

However, optimizing circuits using v_t assignment is usually a post-processing step, and need not be considered during technology mapping.

Gate	Logical Effort	Parasitic Delay
INV	1.00	2.11
NAND2	1.30	3.02
NAND3	1.47	3.94
NAND4	1.61	4.77
NOR2	1.53	3.25
NOR3	1.97	4.13
NOR4	2.42	4.22
AOI21 2	1.88	5.41
AOI21 1	1.41	3.66
AOI22	1.79	9.19
AOI211 2	3.44	9.12
AOI211 1	1.73	4.46
AOI221 2	6.57	14.11
AOI221 1	1.09	3.29
AOI222	2.31	7.62
AOI31 3	1.96	7.82
AOI31 1	1.49	8.31
AOI32 3	1.94	12.65
AOI32 2	2.19	4.76
AOI33	2.13	14.21
OAI21 2	1.81	6.69
OAI21 1	1.49	4.63
OAI22	1.82	6.31
OAI211 2	2.02	8.94
OAI211 1	1.45	4.47
OAI221 2	2.02	9.91
OAI221 1	1.58	6.32
OAI222	1.99	7.94
OAI31 3	2.41	10.36
OAI31 1	1.25	3.98
OAI32 3	2.48	12.86
OAI32 2	1.80	5.02
OAI33	2.42	10.29

Table A.3: The MELTing Library

Bibliography

[AAC$^+$99] D. H. Allen, A. G. Aipperspach, D. T. Cox, N. V. Phan, and S. N. Storino. A 0.2um 1.8V SOI 550MHz 64b PowerPC Microprocessor with Copper Interconnects. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 438–439, February 1999.

[ACG$^+$02] C. Alpert, C. Chu, G. Gandham, M. Hrkic, J. Hu, C. Kashyap, and S. Quay. Simultaneous Driver Sizing and Buffer Insertion Using a Delay Penalty Estimation Technique. In *Proceedings of the IEEE/ACM International Symposium on Physical Design*, pages 104–109, April 2002.

[Ant97] D. A. Antoniadis. SOI CMOS as a Mainstream Low-Power Technology: A Critical Assessment. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 295–300, August 1997.

[ASAGC98] S. J. Abou-Samra, P. A. Aisa, A. Guyot, and B. Courtois. 3D CMOS SOI for High Performance Computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 54–58, August 1998.

[BAC$^+$00] T. C. Buchholtz, A. G. Aipperspach, D. T. Cox, N. V. Phan, S. N. Storino, J. D. Strom, and R. R. Williams. A 660MHz 64b SOI Processor with Cu Interconnects. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 88–89, February 2000.

[BF98] J. L. Burns and J. A. Feldman. C5M - A Control-Logic Layout Synthesis System for High-Performance Microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):14–23, January 1998.

[BKKS98] F. Beeftink, P. Kudva, D. Kung, and L. Stok. Gate-Size Selection for Standard Cell Libraries. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 545–550, November 1998.

[BM97] L. Benini and G. De Micheli. A Survey of Boolean Matching Techniques for Library Binding. *ACM Transactions on Design Automation of Electronic Systems*, 2(3):193–226, July 1997.

[BN02] K. M. Buyuksahin and F. N. Najm. High-Level Area Estimation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 271–274, August 2002.

[BR00] K. Bernstein and N. J. Rohrer. *SOI Circuit Design Concepts*. Kluwer Academic Publishers, Norwell, MA, 2000.

[BVSH02] X. Bai, C. Visweswariah, P. N. Strenski, and D. J. Hathaway. Uncertainty-Aware Circuit Optimization. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 58–63, June 2002.

[CAC$^+$99] M. Canada, C. Akrout, D. Cawthron, J. Corr, S. Geissler, R. Houle, P. Kartschoke, D. Kramer, P. McCormick, N. Rohrer, G. Salem, and L. Warriner. A 580MHz RISC Microprocessor in SOI. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 430–431, February 1999.

[CCH$^+$98] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, and C. W. Wu. JiffyTune: Circuit Optimization Using Time-Domain Sensitivities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1292–1309, December 1998.

[CCW98] C.-P. Chen, C. C. N. Chu, and M. D. F. Wong. Fast and Exact Simultaneous Gate and Wire Sizing by Lagragian Relaxation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 617–624, November 1998.

[CEM$^+$99] A. R. Conn, I. M. Elfadel, W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan. Gradient-Based Optimization of Custom Circuits Using a Static-Timing Formulation. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 452–459, June 1999.

[Cha90] P. K. Chan. Algorithms for Library-Specific Sizing of Combinational Logic. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 353–356, June 1990.

[CHP00] W. Chen, C.-T. Hsieh, and M. Pedram. Simultaneous Gate Sizing and Fanout Optimization. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 374–378, November 2000.

[Chu98] C. T. Chuang. Design Considerations of SOI Digital CMOS VLSI. In *Proceedings of the IEEE International SOI Conference*, pages 5–8, October 1998.

[CK88] H. Y. Chen and Steve S. M. Kang. iCOACH: A Circuit Optimization Aid for CMOS High-Performance Circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 372–375, November 1988.

[CKL$^+$03] Y.-H. Chan, P. Kudva, L. Lacey, G. Northrop, and T. Rosser. Physical Synthesis Methodology for High Performance Microprocessors. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 696–701, June 2003.

[CM01] S. Chakraborty and R. Murgai. Complexity of Minimum-Delay Gate Resizing. In *International Conference on VLSI Design*, pages 425–430, January 2001.

[CP95] K. Chaudhary and M. Pedram. Computing the Area versus Delay Trade-off Curves in Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(12):1480–1489, December 1995.

[CP99] C. T. Chuang and R. Puri. SOI Digital CMOS VLSI - a Design Perspective. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 709–714, 1999.

[CSS$^+$00] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 201–204, May 2000.

[DKS$^+$00] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty. Transformational Placement and Synthesis. In *Proceedings* *of the Design, Automation and Test in Europe Conference*, pages 194–201, March 2000.

[FD85] J. P. Fishburn and A. E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 326–328, November 1985.

[Gal94] L. Gal. Reply to "Comments on the Optimum CMOS Tapered Buffer Problem". *IEEE Journal of Solid-State Circuits*, 29(2):158–159, February 1994.

[GLH$^+$95] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe. A Delay Model for Logic Synthesis of Continuously-Sized Networks. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 458–462, November 1995.

[HJ94] N. Hedenstierna and K. O. Jeppson. Comments on the Optimum CMOS Tapered Buffer Problem. *IEEE Journal of Solid-State Circuits*, 29(2):155–158, February 1994.

[HJLMS03] B. Hu, H. Jiang, Q. Liu, and M. Marek-Sadowska. Synthesis and Placement Flow for Gain-Based Programmable Regular Fabrics. In *Proceedings of the IEEE/ACM International Symposium on Physical Design*, pages 197–203, April 2003.

[HL01] J. Michael Hill and J. Lachman. A 900MHz 2.25MB Cache with On-Chip CPU - Now in Cu SOI. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 176–177, February 2001.

[HP00] Hewlett-Packard. PA-RISC 8x00 Family of Microprocessors With Focus on PA-8700. `http://www.cpus.hp.com/technical_references/PA-8700wp.pdf`, 2000.

[HQGA02] J. Hu, S. T. Quay, G. Gandham, and C. J. Alpert. Buffer Insertion with Adaptive Blockage Avoidance. In *Proceedings of the IEEE/ACM International Symposium on Physical Design*, pages 92–97, April 2002.

[HWKMS03] B. Hu, Y. Watanabe, A. Kondratyev, and M. Marek-Sadowska. Gain-Based Technology Mapping for Discrete-Size Cell Libraries. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 574–579, June 2003.

[IPC94] S. Iman, M. Pedram, and K. Chaudhary. Technology Mapping using Fuzzy Logic. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 333–338, June 1994.

[ITR03] International Technology Roadmap for Semiconductors. `http://public.itrs.net/Files/2003ITRS/Home2003.htm`, 2003. White Paper.

[Jae75] R. C. Jaeger. Comments on "An Optimized Output Stage for MOS Integrated Circuits,". *IEEE Journal of Solid-State Circuits*, SC-10(3):185–186, June 1975.

[JCC97] W. Jin, P. C. H. Chan, and M. Chan. On the Power Dissipation in Dynamic Threshold Silicon-on-Insulator CMOS Inverter. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 247–250, August 1997.

[JOWB00] D.-J. Jongeneel, R. Otten, Y. Watanabe, and R. K. Brayton. Area and Search Space Control for Technology Mapping. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 86–91, June 2000.

[KBS98] Y. Kukimoto, R. K. Brayton, and P. Sawkar. Delay-Optimal Technology Mapping by DAG Covering. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 348–351, June 1998.

[Keu87] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 341–347, June 1987.

[KH99] H. Kapadia and M. A. Horowitz. Using Partitioning to Help Convergence in the Standard-Cell Design Automation Methodology. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 592–597, June 1999.

[KLL82] R. H. Krambeck, C. M. Lee, and H.-F. S. Law. High-Speed Compact Circuits with CMOS. *IEEE Journal of Solid-State Circuits*, SC-17(3):614–619, June 1982.

[KPK$^+$99] Y. W. Kim, S. B. Park, Y. G. Ko, K. I. Kim, I. K. Kim, K. J. Bae, K. W. Lee, J. O. Yu, U. Chung, and K. P. Suh. A 0.25um 600MHz 1.5V SOI 64b

ALPHA Microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 432–433, February 1999.

[KS98] J. B. Kuo and K.-W. Su. *CMOS VLSI Engineering Silicon-on-Insulator (SOI)*. Kluwer Academic Publishers, Norwell, MA, 1998.

[KS01] S. K. Karandikar and S. S. Sapatnekar. Technology Mapping for SOI Domino Logic Incorporating Solutions for the Parasitic Bipolar Effect. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 377–382, June 2001.

[KS03] S. K. Karandikar and S. S. Sapatnekar. Technology Mapping for SOI Domino Logic Incorporating Solutions for the Parasitic Bipolar Effect. *IEEE Transactions on Very Large Scale Integration Systems*, 11(6):1094–1105, December 2003.

[KS04] S. K. Karandikar and S. S. Sapatnekar. Fast Comparisons of Circuit Implementations. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 910–915, February 2004.

[KSD02] P. Kudva, A. Sullivan, and W. Dougherty. Metrics for Structural Logic Synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 551–556, November 2002.

[Kun98] D. S. Kung. A Fast Fanout Optimization Algorithm for Near-Continuous Buffer Libraries. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 352–355, June 1998.

[LCJ$^+$97] P.-F. Lu, C.-T. Chuang, J. Ji, L. F. Wagner, C.-M. Hsieh, J. B. Kuang, L. L.C. Hsu, M. M. Pelella, S.-F. Chu, and C. J. Anderson. Floating-Body Effects in Partially Depleted SOI CMOS Circuits. *IEEE Journal of Solid-State Circuits*, 32(8):1241–1253, August 1997.

[LWGH95] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic Decomposition during Technology Mapping. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 264–271, November 1995.

[LWGH97] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic Decomposition During Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):813–834, August 1997.

[Mag02] Magma Design Automation. Gain Based Synthesis: Speeding RTL to Silicon, 2002.

[MB02] F. Mo and R. K. Brayton. Whirlpool PLAs: A Regular Logic Structure and Their Synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 543–550, November 2002.

[MKA$^+$01] S. Mathew, R. Krishnamurthy, M. Anders, R. Rios, K. Mistry, and K. Soumyanath. Sub-500ps 64b ALUs in 0.18um SOI/Bulk CMOS: Design and Scaling Trends. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 318–319, February 2001.

[NN96] M. Nemani and F. N. Najm. Towards a High-Level Power Estimation Capability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):588–598, June 1996.

[NN98] M. Nemani and F. N. Najm. Delay Estimation of VLSI Circuits from a High-Level View. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 591–594, June 1998.

[PAB$^+$00] S. Posluszny, N. Aoki, D. Boerstler, P. Coulman, S. Dhong, B. Flachs, P. Hofstee, N. Kojima, O. Kwon, K. Lee, D. Meltzer, K. Nowka, J. Park, J. Peter, J. Silberman, O. Takahashi, and P. Villarrubia. "Timing Closure by Design," A High Frequency Microprocessor Design Methodology. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 712–717, June 2000.

[PBR96] R. Puri, A. Bjorksten, and T. E. Rosser. Logic Optimization by Output Phase Assignment in Dynamic Logic Synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 2–6, November 1996.

[PC98] R. Puri and C. T. Chuang. Hysteresis Effect in Pass-Transistor Based Partially-Depleted SOI CMOS Circuits. In *Proceedings of the IEEE International SOI Conference*, pages 103–104, October 1998.

[PSS$^+$03] L. Pileggi, H. Schmit, A. J. Strojwas, P. Gopalakrishnan, V. Kheterpal, A. Koorapaty, C. Patel, V. Rovner, and K. Y. Tong. Exploring Regular Fabrics to Optimize the Performance-Cost Trade-Off. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 781–787, June 2003.

[RAPS99] P. Rezvani, A. H. Ajami, M. Pedram, and H. Savoj. LEOPARD: A Logical Effort-based fanout OPtimizer for ARea and Delay. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 516–519, November 1999.

[RBB99] S. Roy, K. Belkhale, and P. Banerjee. An α-Approximate Algorithm for Delay-Constraint Technology Mapping. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 367–372, June 1999.

[SIS99] L. Stok, M. A. Iyer, and A. J. Sullivan. Wavefront Technology Mapping. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 531–536, March 1999.

[SK99] K. L. Shepard and D.-J. Kim. Body-Voltage Estimation in Digital PD-SOI Circuits and Its Application to Static Timing Analysis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 531–538, November 1999.

[SK01] K. L. Shepard and D.-J. Kim. Body-Voltage Estimation in Digital PD-SOI Circuits and Its Application to Static Timing Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(7):888–901, July 2001.

[SKB$^+$96] L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler. BooleDozer: Logic Synthesis for ASICs. *IBM Journal of Research and Development*, 40(4):407–430, 1996.

[SMCK01] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick. The Scaling Challenge: Can Correct-by-Construction Design Help? In *Proceedings of the IEEE/ACM International Symposium on Physical Design*, pages 51–58, April 2001.

[SMCS03] A. Srivastava, S. O. Memik, B.-K. Choi, and M. Sarrafzadeh. Achieving Design Closure Through Delay Relaxation Parameter. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 54–57, November 2003.

[SNvGK02] K. Sulimma, I. Neumann, L. P. P. P. van Ginneken, and W. Kunz. Improving Placement under the Constant Delay Model. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 677–682, March 2002.

[SRLB97] G. Swamy, S. Rajamani, C. Lennard, and R. K. Brayton. Minimal Logic Re-Synthesis for Engineering Change. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 3, pages 1596–1599, June 1997.

[SRVK93] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S.-M. Kang. An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(11):1621–1634, November 1993.

[SS91] R. F. Sproull and I. E. Sutherland. Theory of Logical Effort: Designing for Speed on the Back of an Envelope. In *IEEE Advanced Research in VLSI*, pages 1–16, 1991.

[SSH99] I. Sutherland, R. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, San Fransisco, CA, 1999.

[SSL$^+$92] E. M. Sentovich, K. Jit Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[SSP02] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi. Fast and Exact Transistor Sizing Based on Iterative Relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(5):568–581, May 2002.

[STMTS00] D. Stasieak, J. Tran, F. Mounes-Toussi, and S. Storino. A 2nd generation 440ps SOI 64b Adder. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 288–289, February 2000.

[TCT$^+$98] C. Tretz, C. T. Chuang, L. Terman, M. Pelella, and C. Zukowski. Performance Comparison of Differential Static CMOS Circuit Topologies in SOI Technology. In *Proceedings of the IEEE International SOI Conference*, pages 123–124, October 1998.

[TCW97] Y.-C. Tseng, S. C. Chin, and J. C. S. Woo. The Impact of SOI MOSFETs on Low Power Digital Circuits. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 243–246, August 1997.

[TMBW90] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance-Oriented Technology Mapping. In *Proceedings of the 6th MIT Conf. on Advanced Research in VLSI*, pages 79–97, 1990.

[vG90] L. P. P. P. van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 865–868, May 1990.

[Vis97] C. Visweswariah. Optimization Techniques for High-Performance Digital Circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 198–207, November 1997.

[VS02] M Vujkovic and C. Sechen. Optimized Power-Delay Curve Generation for Standard Cell ICs. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 387–394, November 2002.

[WSA98] A. Wei, M. J. Sherony, and D. A. Antoniadis. Effect of Floating-Body Charge on SOI MOSFET Design. *IEEE Transactions on Electron Devices*, 45(2):430–438, February 1998.

[XS96] W. Xiaoqing and K. K. Saluja. A New Method Towards Achieving Global Optimality in Technology Mapping. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 9–12, November 1996.

[ZS98] M. Zhao and S. S. Sapatnekar. Technology Mapping for Domino Logic. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 248–251, November 1998.